大话移动APP测试 Android与iOS

应用测试指南

陈晔 / 著
行业ID：Monkey

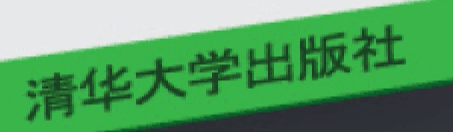
讲述技术，更揭示测试行业现状和内幕。

无论你身处互联网或移动互联网，这本书都值得你去阅读。



讲述移动互联网测试行业的动荡血泪史。所有内容均亲历亲为，谢绝道听途说。





清华大学出版社


●●●○○ AT&T
11:56 AM
大话
移动APP测试
Android与iOS
应用测试指南
陈晔／著
行业ID：Monkey
清华大学出版社
北 京
Mailboxes
Inbox
Edit

内 容 简 介

移动互联网软件测试无论从思想还是技术上都与传统互联网产品或软件产品截然不同，导致了很多正在移动互联网中摸索的测试人员迷失了方向。作为一名移动互联网的测试从业人员，需要正确的三观、强大的“武功招式”（测试技术）和雄厚的“内力”（更快的学习能力），而本书恰到好处地结合了这三点。

本书内容包含：移动互联网测试人员的面试，用户体验测试，功能测试，常用测试工具，常用框架，APP 测试案例，以及更多的从业相关思维、手段等非技术内容。

本书并非纯技术书籍，但可以说是移动互联网测试、甚至是所有测试工程师必读的一本书籍。如果你在测试行业迷了路，本书可以为你指出一条属于你自己的正确道路；如果你初入移动互联网，那么本书可以帮助你快速融入这个新兴行业，并全面了解和掌握这个行业所需要的技术和方法；如果你是一名老兵，那么本书同样可以起到温故而知新的作用，同时会说出你在测试行业中不曾说出的那些心里话；如果你是一名移动互联网行业任意职位的从业人员，你可以了解 APP 测试在整个产品开发中的位置和重要性，并在工作中与测试人员紧密配合，同时给予这个职位应有的尊重。

图书在版编目(CIP)数据

大话移动 APP 测试：Android 与 iOS 应用测试指南 / 陈晔 著. —北京：清华大学出版社，2014
ISBN 978-7-302-36879-3

Ⅰ.①大…　Ⅱ. ①陈…　Ⅲ. ①移动终端—应用程序—程序测试—指南　Ⅳ.①TN929.53-62

中国版本图书馆 CIP 数据核字（2014）第 131365 号

责任编辑：栾大成
责任校对：徐俊伟
责任印制：
出版发行：清华大学出版社

网　　址：http://www.tup.com.cn，　http://www.wqbook.com
地　　址：北京清华大学学研大厦 A 座　　　**邮　　编：**100084
社 总 机：010-62770175　　　**邮　　购：**010-62786544
投稿与读者服务：010-62776969, c-service@tup.tsinghua.edu.cn
质量反馈：010-62772015, zhiliang@tup.tsinghua.edu.cn

印 刷 者：
装 订 者：
经　　销：全国新华书店
开　　本：188mm×230mm　　**印　　张：**19.25　　**插页：**1　　**字　　数：**363 千字
版　　次：2014 年 8 月第 1 版　　　**印　　次：**2014 年 8 月第 1 次印刷
印　　数：1~4000
定　　价：59.00 元

产品编号：054476-01

前　言

“招聘之前自己先体验一下这个岗位，这不仅有助于找到合适的人，还会帮助你日后更好的管理”。——《Rework》

如果大家看到现在这句话，说明我人生第一本书已经顺利出版了。上面这句话是我在 2013 年读过的少数几本书中最有感触的一句话。其实，不仅对招聘是这样，所有事情都应该如此。

在写前言之前，我大概谷歌了下前言应该写什么内容，接下来，我就说说这本书的目标群体，缘由以及想达到的目的。

先来说说目标群体吧，免得很多人看完了说浪费他的时间。我并不想说本书所阐述的文字和观点多么的正确，但与测试行业的其他书籍有着天壤之别的地方就是书中阐述的都是一个一个非常真实的例子和项目。



提示：有很多人听到我说正在写书会说“哇！好厉害！都能出书了！”，这反应就如同看到一个××大学毕业，在×××公司做了很多年，有着×××头衔的“大佬”后，发出的惊叹一样。但我想说的是，在发出惊叹之前，你们真的认识这个“大佬”吗？很熟吗？还是仅仅为这些头衔所忽悠了。现在这个社会里，出个书根本不是什么难事，关键在于出书的初衷是什么，是否真的有自己的想法。太多的“大佬”和“教授”出书和搞沙龙说白了就是为钱，谁都爱钱，但是为了钱扯淡忽悠就是你的不是了。综上所述，本书理论派和忽悠为生者勿扰即可，我在这里不多作解释，认识我的人都很清楚我指的是哪些人群。

说到写这本书的缘由是一个比较长的故事了，喜欢看故事的读者可以在本书的后记中读到，不喜欢的可以无视并直接挑选你喜欢的章节狠狠地给我提建议。

看这本书就像带你去品尝一顿由我精心制作的大餐。或许你第一次吃、或许你曾经吃过类似的食物，无论如何我相信你都会有全新的味觉体验。这套大餐分别由前菜、主

菜、甜点和餐后水果组成。

前菜由第 1～3 组成，让大家品尝移动互联网测试行业面试、行业的现状以及特别需要关注的“用户体验测试”等菜色。

- 测试面试——结合我经历过的面试（包括面试别人以及被别人面试的经历）所看到的现象，从中抽取了部分具有代表性的例子进行描述，也包括一些自己的感受。
- 行业现状——相信品尝之后会有涩涩的味道，行业现状并没有你看到或者想像的那么美好，但这就是事实（就如我之前说的，我所描述的并非正确，但一定是事实。依然是那句话，一切眼见为实，不要盲目的去相信任何人，尤其是自己为自己套上很多高帽子的人。）
- 用户体验——从 Android 和 iOS 两者的设计区别入手，让大家慢慢地了解各种应用在用户体验上的细微差别。现在是大数据时代，产品在用户体验上的设计进步，其背后依托的正是这些会说话的数据。

无论你是否已经深陷移动互联网的泥潭，我相信前菜非常的可口的，大家可以像读故事一般获取第一手信息，就目前来看我觉得这些信息还是很新鲜（接地气）的。

接着介绍主菜，由第 4～8 章组成，让大家品尝到移动互联网测试行业中的应用功能测试要点、常用工具、常用框架、实际项目案例以及性能测试等菜色。

- 功能测试要点——从移动应用测试的日常工作中总结出相对优先级较高的测试切入点，用我的话来讲都是真正的经验之谈，没啥技术含量。
- 常用工具——介绍及实践日常工作中经常用到的移动应用测试工具。品尝过后应属最美味的一道菜肴。
- 常用框架——介绍及实践一些常用的框架，无论是分层测试还是自动化测试都会涉及到对这些框架的检测。
- 实际项目的案例——整个章节详细介绍了两个实际项目案例的测试过程，属于对前面几章的综合实践。

- 性能测试——介绍及实践移动应用的性能测试方法，包括工具的使用以及一些测试思路，我对性能也了解有限，故味道应该很淡。

如果你的牙口不好、肠胃不佳，主菜也许难以咀嚼或者一下子消化，这不是问题，可以慢慢品尝，不必急着一天拿下，不是么?

接下来是甜点时间，甜点是主菜之后的奖赏，可以使得吃的人放松，由附录 A 和附录 B 组成。先给大家祝兴添点“乐子”，对很多自称测试工程师却根本不知道什么是“测试”的朋友以及测试行业经常出现的问题进行吐槽（这不由得让我想到业内相当出名的槽神（@槽神刘叫兽））。吐槽本身是很轻松的事，小吐怡情，何乐而不为呢？之后，公平起见，大家再来吐槽我的观点。

- 测试人员的自我修养——整篇吐槽不知道“测试”为何物的“测试人”，业界的“忽悠大佬们”就不用提了，他们已经没有任何修养可言了。
- 测试行业常见问题——整篇吐槽业内几乎每天都会出现的问题。也许很多人会觉得这些问题都很初级，我可不这么看，俗话说“大象怕老鼠”，多少人多少次倒霉就倒霉在这些初级问题上，说谁谁知道。

我看到你一直在撇嘴，是不是感觉甜的要掉牙了或者都不屑于吐槽我了？这就差不多了，“不吐不快”嘛，吐净了就快入道了——我们需要不停地以人为鉴，反思自己，否则被人吐槽还是好的。

最后是餐后水果部分，附录 C 摘抄了我自己博客中觉得有价值的几篇，有助于饭后消化。

好吧，我还是说实话吧，其实甜点和水果并不在我最初的菜单中，只是这 8 个月的时间洋洋洒洒写下来，觉得有必要增加这样两章内容。也许你觉得是凑数的，但在我自己看来，这两章阐述的观点也是我心中认为本书原本就该拥有的部分。

最后说一下这本书的目的——让更多的人能够看到我分享的经验。

在编写本书过程中最大的难点有两个：第一点是撰写。我从 2013 年 4 月份开始撰写本书，女儿刚出生不到一个月，不得不说这一段日子非常难熬，拼的是体力和毅力。

正因为处在这样一个特殊的时期，我更能自豪并负责地告诉大家，书中的每一个字都是自己三年多工作经验的真实提炼，为的只是让大家看到我的分享，将自己的所见所感传达给更多的人，并与行业其他忽悠扯淡的书划清界限。

第二点是审核。在这个行业中，技术大牛的人的确很多，但我思前想后最终没有找到适合审核全书的人，这一点大家浏览过全书后就会发现。不过，这里依然要感谢参与审核部分章节的朋友——张立华（@晋恒温）、谢琼（@谢小麦啊）、张昊翔（@美满的张昊翔）和胡明伟（@ Mingway_Hu），他们分别从不同的角度做了审核。做过审核的同学都会知道，审核在某种意义上来讲比撰写更累人，在此非常感谢我的小伙伴们。

前言是在我写完整本书之后才回来补充的，这道大餐背后隐藏着太多的酸甜苦辣。或许你之前就认识我，或许你根本就不知道我是谁，又或许我已经被你归了“扯淡”的行列，但是，无论如何，我在这里恳请你能够读完整本书（包括后记），再来评价，至少在测试和原则上。如果你认同我的观点和原则，那么让我们一起为了测试行业的进步而奋斗吧！

陈晔

目　录



第 1 章　移动互联网面试

可能很多人觉得可笑，为什么要花一章来讲面试？莫非是凑字数？面试这项活动成败与否不确定因素实在太多，需要一定的运气，也需要一定经验和技术积累。在我们长期稳定的工作中，面对的一直是同事、客户、老板这样一群固定的人。慢慢地，我们就失去了与陌生人打交道的能力，失去了结交新朋友的感觉。面试其实是一个长期行为，并非只是为了找工作才需要去面试。

面试从表面上来看是一个人在考核另外一个人，但是实质上，面试是一个非常复杂的过程。

1.1 为什么要面试

面试是与陌生人打交道的过程，一般来说，人在世界上看到最多的生物就是人，而最难接触、最难相处的也是人。面试的时候，大多数人都会非常紧张，其中一部分原因就来自于坐在自己对面的是一个陌生人。紧张的情绪会让应聘者四肢冰冷，额头出汗，更多的是语无伦次，表达不清。从面试官角度来讲，他心里很清楚应聘者多少都会有紧张的情绪，但作为测试工程师这样一个职务，要求在将来的工作中更多的是必须和项目或者非项目的同事合作。一个与陌生人打交道非常紧张的人并不一定会影响到接下来的测试工作，但是一定会影响到沟通的效率，因此面试官自然会考察这一点。

面试是一个沟通的过程。面试的过程中会被问到很多问题，包括工作、生活、技术、人生等。如何快速、正确地理解面试官的问题以及简短、准确地描述自己的观点就变得极其重要。很多应聘者为了能在人群中脱颖而出故意表现自己的能说会道，这种行为往往适得其反。从心理学角度来讲，人在多说话的时候情绪容易亢奋，更不容易发现自己话中的错误。站在企业和面试官的立场，他们需要的是一个满足他们需求的人，而不是只会耍嘴皮子的人。应聘者的理解能力和表达能力将会大大影响以后和同事的合作关系以及个人发展。

面试是一个学习的过程。在这个过程中，无论应聘者成功与否，都能够发现自己的一些缺陷。这种缺陷可能是技术上的、思想上的、心理上的，等等。当然，这种学习并非在短短几个小时内就会完成，更多的是需要应聘者对每次面试进行总结，从而弥补自身不足之处。很多人经过多次面试后，发现自己总在某些方面栽跟头，其实就是自己每次面试后都持消极沮丧的态度，不去总结的后果。

1.2 面试的流程

对测试人员的面试流程一般分成 4 部分。

- 第 1 部分：大部分的外企在这一部分是让应聘者做笔试。笔试内容一般都会包括编程类、算法类、软件工程类、数据库、翻译类、智力问答类等等，可以说是应有尽有。

- 第 2 部分：应聘者和面试官互相寒暄，应聘者做自我介绍。
- 第 3 部分：回答面试官的问题，提问的点可能是根据应聘者的自我介绍即兴提出，也可能是预先设定的面试问题等等。
- 第 4 部分：应聘者和企业人事之间的交流。

一般流程不外乎以上的模式。应聘者往往最头疼的就是碰见笔试和面试车轮战。但是其实这种流程是存在很大的弊端，在后面的小节中会提到。

1.3　面试的频率

读者也许会很奇怪，面试还有什么频率？难道不是为跳槽找工作才去面试吗？在其他行业可能是这样，但是在移动互联网做测试就得跳出这个固定模式。从移动互联网行业的发展来看，其运用到的技术以及思想在短短的几年内有了翻天覆地的变化。定期的面试能够让测试人员积累面试经验，变得不再害怕面试，也更能够抓准现在行业中企业真正关注应聘者的什么能力。当然，这里所指的定期面试不是唆使测试人员定期跳槽，而是抱着学习和积累经验的态度前往面试。

1.4　面试实战

小陈同学是一名应届生，某天他拿着自己的简历前往某企业进行面试。我们先来看一下这位同学的简历。

个人简历

个人信息:

姓　　名:	小陈	性别:	男
手机号码:	18621519900	年龄:	26
电子邮件:	snowangelsimon@gmail.com	所在地:	上海
婚姻状况:	已婚	工作年限:	4
教育程度:	专科		

职业意向:

期望行业: 计算机软件
期望职位: 软件测试工程师
期望地点: 上海

自我评价:

为人诚恳上进，善于不断学习充实自己，适应环境能力强，用于接受挑战，能承受较大的工作压力。

特长与专业技能:

熟练软件测试工作流程，掌握测试用例设计，测试执行和缺陷跟踪
熟悉运用多种方法进行黑盒测试，如：等价类、边界值、因果图等
对于软件测试有浓厚的兴趣

Java	12 个月
C	24 个月

教育/培训经历

XXX 培训机构进行了软件测试培训，完成了所有的课程

面试正式开始了，下面模拟了一些可能出现的场景。

面试官：小陈同学你好，请先自我介绍一下。

小陈：您好，我叫小陈。我是 xxx 大学计算机专业毕业的学生。我在学校学了 xx 课程，担任了学校 xx 职务。我毕业之后想找一份和计算机有关的工作，但觉得自己实战经验可能不够，所以前往了 xx 培训了 xx 个月，目前已经毕业。主要学习的技术有软件测试的基础，设计方法以及各种编程语言。

分析面试官心理：

可能是：面试的常规流程应该从应聘者的自我介绍开始

可能是：我忘记看应聘者简历了，通过自我介绍我了解一下吧

可能是：从应聘者的自我介绍中观察应聘者对自身的了解，并了解应聘者以前的工作经历。

从面试官角度来讲，让应聘者自我介绍是一种缓和气氛、以便让应聘者逐步进入自己的最佳状态的一种手段。（当然，也许一些读者会觉得自己根本不会自我介绍，或者一开始自我介绍就紧张，这种情况的确很常见，但我只能说这是你一定要克服的问题。）

应聘者注意点：

小陈同学这里所作的自我介绍中规中矩，但没有什么特色。很多应聘者面试之所以失败往往从自我介绍就开始了。按照常规流程一般是姓名、所在企业、以前做过什么项目等等的介绍。这样并没有错，是一种很保守的介绍方式。在这里应聘者需要注意的是四个字“扬长避短”，不要为了让别人认为自己很能干、涉及面很广而夸大其词。应聘者要学会将自己的长处完完全全的体现在对方面前，并让对方跟着自己的思路。这样的话，面试官往往会问一些相关的问题，应聘者也能够因为有所准备而从容应对了。

面试官：小陈同学，请问你是否了解 Android，iOS 或者其他平台的优缺点呢？请简单描述一下。

小陈：您说的这些以前我都没有什么经验，刚刚毕业而已。

这里小陈犯了一个很大的错误，作为一个应届生来讲这样的回答等同于告诉面试官“我在学校除了上课都在浪费时间，我没有关心过任何相关的信息”。所以无论面试官问了什么，都不要这样去回答。面试官自然知道来面试的人是否刚刚毕业。他问一些问题

主要是看应聘者以前有没有一些积累，从中看出这个人的习惯。所以，假设一个人在学校的时候就保持着自我学习和探索精神，那么这个习惯在工作中也会生效。反之，对于面试官来讲这只是一种推脱之词。

就这样，面试的一个小时结束了，其结果我想大家都心中有数：小陈同学没有通过这家企业的面试。不过小陈同学很不甘心，他决定工作几年之后再向这家企业发出挑战。以下是小陈同学 2 年后的简历。

个人简历

个人信息：

姓名：	小陈	性别：	男
手机号码：	18621519900	年龄：	26
电子邮件：	snowangelsimon@gmail.com	所在地：	上海
婚姻状况：	已婚	工作年限：	4 年
教育程度：	专科		

职业意向：

期望行业： 计算机软件
期望职位： 软件测试工程师
期望地点： 上海

自我评价：

为人诚恳上进，善于不断学习充实自己，适应环境能力强，用于接受挑战，能承受较大的工作压力。

特长与专业技能：

熟练掌握 Android，iOS 测试工具的使用
熟练掌握了 Android 黑盒功能测试，模拟用户各种使用场景
会使用 Monkey 工具进行测试

Java	12 个月
C	24 个月

教育/培训经历

XXX 培训机构进行了软件测试培训，完成了所有的课程

面试又开始了，以下是模拟的场景：

面试官：小陈，你对于移动互联网测试的自动化框架以及工具了解多少？请描述一下吧。

小陈：我以前是做 Android 测试的，一直是手工测试。所以这方面可能了解不多。

可能广大的移动互联网测试人员看到小陈的描述会认为自己肯定不会这么说。但是从事实上来讲，的确有很大一部分的应聘者就是这样说的。原因出自两点：其一，自己说习惯了，事实也是一直在做功能测试；其二，自己内心的自卑或谦虚，加上面试的紧张，一时脱口而出。

小陈这样描述等于在告诉对方“我对此很没有自信”。试问自己对自己都没有足够的信心的话，如何让企业和面试官对你有信心呢？从另外一个角度来看，作为从事软件测试的人，不去涉猎一点自动化测试，这本身就是不够上进、满足现状、不爱学习的体现。软件测试是一个技术岗位，理应对行业的新技术有所涉猎才对。此外，应聘者如果这样描述，说明他作为一个软件测试人员根本没有了解到软件测试的核心。软件测试的核心是测试用例的设计，用例设计的好坏在一定程度上体现了一名测试人员的能力以及价值。而是否用手工或者别的方法来完成这些用例测试只是使用方法不同而已，所以千万不要因为手工测试而看低自己，觉得手工测试没有技术含量或者没有前途。要知道，软件测试具备的能力不是只有手工和自动化测试这两项，还有许许多多的东西需要我们去学习。这些在后面的章节会详细解释。

面试官：能否描述一下 Android SDK 文档中提到的测试工具呢？在项目中你是怎么使用的？

小陈：以前的工作忙于业务，并没有仔细看过相关的开源文档。

面试官：从简历上看你有一定编程语言基础，请用你熟悉的语言写一个冒泡排序吧？

小陈拿过纸笔，最终还是没有写下任何一行代码。

在实际的面试中，面试官更可能问一些具体的问题，比如：请问 Monkey 怎么使用？或者对 instrumentation 是否了解？……但就上面举的例子来说，很多测试人员并不知道

该怎么回答。例如有的应聘者会说没有看过，或者说知道几个工具但是没怎么用过等等。

一般而言，测试人员不知道怎么回答的原因有 3 个：

（1）完全不知道存在这份开源文档；

（2）文档是英语编写的，看不懂；

（3）自己所在的环境和项目根本用不到文档中提到的工具或技术。

无论出于哪种原因，结果只有一个，就是应聘者不够了解相关技术，并且从侧面也可以看出应聘者没有打破沙锅问到底的钻研精神，而这点对于一个测试人员来讲是致命的不足之处。如果你想学习移动互联网测试，如果你在工作的过程中对某些技术不了解，那么开源文档永远是你的不二选择。不仅仅 Android 如此，别的任何一项技术也是如此。

我就看到过许多应聘者在面试过程中不懂得扬长避短，应聘者在没有扎实的 C++、Java 功底或者 Android 软件测试经验的情况下，随意在简历或者面试中去夸大这些经验和技能，结果却偏偏暴露了自己的短板。这样的情况比比皆是，希望广大测试人员要实事求是。

1.5 如何正确地招聘测试人员

前几个小节是写给应聘的测试人员看的，这一节是写给广大企业的面试官看的。

面试的时候应该怎样去考核一个测试人员呢？与考核一个开发人员一样吗？错！请丢弃那些无用的笔试题以及那些已经用烂了的技术开发题吧。你们需要从以下几点去了解应聘者，选出真正的人才。

第一，应聘者从事测试工作的原因。

每个选择软件测试这份职业的人都有自己的原因，这个原因在很大程度上决定着他可以做多久，是否踏实。这样重要的问题在作者曾经经历的面试中几乎很少被问到，所以，当我在招聘测试人员的时候，首先就会询问该问题。得到的往往是以下几种答案：

- 我只想做和计算机有关的工作。
- 我其实是想做开发工程师的，但是一直通不过面试，所以选择做测试工程师。
- 我在×××培训机构培训的就是测试，我觉得自己很合适。
- 我是一个女生，我觉得女生做测试比较合适。
- ……

以上任何一个答案都不具说服力，希望广大的软件测试工程师们扪心自问：你们做这份工作的初衷到底是什么？为什么现在还在继续做？也许有人会觉得自己比较细心或者是女生就比较合适，从短期来看细心或许能够支撑一个人做一件事情，但要是做十年二十年甚至更久呢？试问细心等还算是支撑的理由吗？

第二，应聘者测试用例的设计能力。

很多面试官可能会问与此相关的问题，但是有的太专注业务方向，应聘者如果没有相关领域的经验恐怕很难设计出有深度的用例；有的却是太过于形式化，应聘者随意说了两个“等价类，边界值”就可以蒙混过关。用例设计的基础是无二义性以及具体的实现步骤，只有应聘者在面试过程中根据具体的问题写出具体用例的情况下才有考核的价值。



在我面试别人的过程中，往往会用以下几道题目来考核应聘者的设计以及思维能力，这里我想分享一下，先看第 1 题。

假设有如右图所示的这样一个登陆框，请问你该如何进行测试用例的设计？

回答此类问题的加分项有：

- 进行需求的确认，包括输入框的输入类型或者长度限制等等。

- 设计用例思路清晰有条理，而不是脱口而出、想到什么说什么。
- 进行具体的用例设计，而不是简单地说“等价类”、“边界值”等。
- 从整体去考虑问题，比如单击按钮之后的反馈，程序所在的系统有什么特性以及一些并发情况的发生。
- 从用户体验角度出发，比如密码框输入之后是否为暗文显示以及交互性是否友好等。

当然考核项不仅仅只有以上这些，但至少是我在面试中会评判应聘者的指标。下面我们看第 2 题。



假设有右图所示这样一个应用，共有 3 个输入框，该应用的功能是输出用户输入日期的下一天。

此类问题有一部分的考核点和第一题相似，不过也有特定的加分项：

1）遇到特殊年份的逻辑跳转，比如闰年。

2）遇到特殊月份的逻辑跳转，比如 2 月。

3）遇到日期跨月、跨年的逻辑跳转，比如 7 月 31 日跳转到 8 月 1 日，2000 年 12 月 31 日跳转到 2001 年 1 月 1 日等。

4）当日期显示位数有变化时的逻辑跳转，比如月份从 9 变化成 10 的时候。

5）同样的，考核点不仅仅只有以上这些，但是这个问题相比第 1 题来讲，更多的是考察测试人员对于一个产品的理解力以及用例设计能力。

第 3 题是一道纯粹考核测试人员逻辑思维题，题目如下：

假设一共有 25 匹马，一个赛场。赛场有 5 个赛道，就是说最多同时可以容纳 5 匹马一起比赛。假设每匹马都跑得很稳定，不使用任何其他工具，只通过马与马之间的比赛，试问，最少得比多少场才能知道跑得最快的 5 匹马?

其实回答这道题目本身并不困难，重要的是考核思考的切入点。这里给出一个提示。大家可自行解出答案。

我们可以将 25 匹马分成 5 组，分别如下所示：

1 组	A1	A2	A3	A4	A5
2 组	B1	B2	B3	B4	B5
3 组	C1	C2	C3	C4	C5
4 组	D1	D2	D3	D4	D5
5 组	E1	E2	E3	E4	E5

假设我们将 25 匹马分成 5 组，经过 5 场比赛之后，假设第 1 组的排名是 A1>A2>A3>A4>A5，其他几组依此类推。我们将每组的第 1 名组成一组再进行比赛，假设结果是 A1>B1>C1>D1>E1，那么 A1 为第 1 名，第 2 名不是 A2 就是 B1。大家可以按照这样的思路去思考。

第 4 题（一般也就是最后一题）虽然是道老掉牙的题目，但是同样能够看出测试人员的设计能力，题目如下：

假设让你去测试一部经常出现在地铁站或者企业中的那种投币自动售货机，怎么设计测试用例。

该题目考核点比较多，也很散，不过主要抓住以下几点就能加分：

- 条理清晰，会通过正常情况、非正常情况详细地设计用例。
- 能够考虑到各种特殊情况，包括断电、漏水、缺货等。
- 能从用户体验角度考虑，比如出货的温度、时间、找零等。
- 能够从用户群体考虑，比如小孩、残障人士等。

这道题目是十几年前面试常用的题型，我一直喜欢使用，因为这种题目有它开放性的一面，所以它的答案也很多样化，能够看出一个人对于一件事物是否有自己新的认识。

第三，考核应聘者作为测试人员的基本素质。

在面试的问答过程中很容易了解应聘者作为测试人员的基本素质。例如当应聘者面对一个问题的时候，是否会本能地问清楚问题的需求，沟通上是否存在障碍等等。

第四，考核应聘者的专业技术能力。

软件测试是一个技术职位，其技术考核自然不可缺少。面试官应该根据自己企业、自己团队的实际需要进行技术考核，不要以面盖点，从而错误的去认识应聘者的能力。

第五，考核应聘者的态度。

这里所指的不仅仅是工作态度，也包括生活态度、学习态度等等。这些态度可能不是通过问答能够考核的，更多的是贯穿在整个面试过程中，面试官对于应聘者的总体感觉。态度是一个测试人员能够在这个行业走多远、走多深的决定性因素。

1.6 移动互联网测试招聘现状

这里所描述的现状，时间截至到作者写这本书为止。下面列出了一份我们常见的移动软件测试的 JD：

职位职能：软件测试
职能描述：
1、 指定、编写软件测试方案与计划；
2、 按时完成软件测试工作任务；
3、 编写测试文档、测试报告、提交测试结果；
4、 改进软件测试流程、工具和质量；
5、 参与测试结果评审。

岗位要求：
1、3 年以上测试工作经验，有管理软件测试经验者优先；
2、熟悉 JIRA、QC、LoadRunner 等测试工具，有 1 年以上 iOS 或 Android 测试工作经验者优先；
3、能够熟练书写测试脚本、测试用例，熟练使用各种测试工具，具备软件测试环境搭建的能力；
4、了解软件工程思想和方法，了解基本数据库系统及网络知识；
5、较强的发现问题、分析问题的能力；较强的语言表达能力和文档撰写能力；
6、工作责任心强，细致，耐心。

整个移动互联网测试的就业现状出现了一种很奇特的现象：应聘需求量非常大，但是应聘者却不知道以什么样的技能和能力去应聘；相应的，招聘需求量也很大，但是从千篇一律的招聘简章（如上图）中看得出来，大部分的企业和面试官自己都不清楚自己需要什么样的测试人员。这种情况直接导致了面试过程的不规范性，面试官盲目地考核应聘者。很多企业面试测试人员仅仅就是面试编程和算法，而当应聘者问及自己到企业具体做什么的时候，也往往都得不到确切的答案。

曾经有一个刚面试完的测试从业人员问我一个问题："我想找份自己理想的测试工作，但是经过几次面试之后，我迷茫了。你能否告诉我，我到底要学什么？"。其实我认识他也很久了，对他问出这样的问题我既感兴趣又觉得难过，问其原因，得到答案是这样的："其实开始我觉得自己还清楚要学什么，但是经过几次面试，看到很长很长的笔试题以及各种和软件测试好像毫无关系的面试题之后，我开始觉得自己什么都欠缺，开始怀疑自己。"其实我听到这样的回答并不惊讶，因为自己看到过太多这样的情况。

我见过很多测试人员换工作的理由是一直做手工测试，想学习自动化测试（当然隐含着待遇差的意思）。从短期来看，换工作可能会解决他们的问题。但是从长远来看，到任何一家企业，只要时间够长，我们总会觉得到了一种学习上的瓶颈，觉得没有进步，都是自己熟悉到甚至有些厌烦的东西，所以换工作其实并不能真正解决问题。一名测试人员如果要想有长期和系统的提升，就必须不能被自己所在的企业、团队、项目等外在因素所制约，要学会自己定目标，不停地学习才是真正的解决之道。

1.7　大公司和小公司

很多应届生或测试从业人员在面临这样的问题的时候往往不知道该怎么做：眼前放着一家知名大型企业和一家刚创业的小公司，那么你会选择哪一家？在我主持的面试中，遇见过大大小小的企业出来的测试人员，大家看了下面的故事之后，可能就会有自己的答案了。

1.7.1 故事一

王小姐，所在企业：某知名互联网企业，职位：软件测试工程师

在我拿到她简历的时候其实心里还是很高兴的，因为看到简历上满满的项目经历，又是在知名互联网企业，对我自己所在的创业公司来讲是难能可贵的。所以我满心欢喜地开始了我的面试。部分对话如下：

我：请问以前项目测试用例都是你们自己编写的吗？

王小姐：由于我们公司人很多，项目环节分的很细，所以编写测试用例是由另外一个组的测试人员负责，不是我们自己写的。

我：那么你们平时除了执行测试用例之外，还做点别的什么类型的测试吗？

王小姐：使用一些工具进行自动化测试。

我：这些工具你有没有了解过是怎么实现的，或者运行原理是什么呢？

王小姐：没有关注，只是去使用，进行产品测试。

我：你为什么离职呢？

王小姐：主要觉得工作太枯燥，没有学习和进步空间，都做了好几年了。

从上面的对话当中不难看出，虽然王小姐有着不错的工作背景，但是其实虚的很，我想大多数的面试官无法接受她。移动互联网企业近几年开始越来越看重实际能力，学历或者企业背景已经没有以前那么大的比重了。自然，王小姐没有通过我的面试。

如果你有机会进入一家知名大企业工作，那么绝对是一个不错的机会。你可能能够加入一个很好的团队，参与一个很好的项目；可能有很积极的环境，认识很多技术厉害的同事等等。你得好好利用这些资源，这些都是能够让自己的能力在短时间内得到巨大的提升的机会。如果有这么好的环境，你还在混日子的话，那只能说你已经自我放弃了，更别提指望得到面试官的青睐了。

1.7.2　故事二

张女士，所在企业：某移动互联网创业公司，职位：高级软件测试工程师

这份简历也是事先让我比较看好的，原因在于张女士的背景肯定能够让她更快地适应创业公司的氛围，并且从职位上来看能力应该也是不错的。于是我高兴地开始面试。部分对话如下：

我：张女士，请问平时你怎么进行 Android 的测试呢？测试一个产品的周期大概需要多久呢？

张女士：我以前在创业公司，公司就我一个测试人员。测试用例、文档都是我自己写的。平时主要都是功能测试，自动化测试自己也在研究。目前只是执行一下 Android 的 MonkeyTest。测试产品的话一个月左右发布一次新版本。

我：张女士，我想问一下，Monkey 工具具体是怎么执行的？会加哪些参数？你是怎么学习的呢？

张女士：就是一句脚本执行一下，每次执行几万次的样子，并没有增加什么特定的参数。

我：请问是否仔细看过 Google 的 Android 开源文档呢？

张女士：好像都是公司里的开发人员在看，我自己没有怎么仔细看，英语不是很好。

从上面的对话不难看出，张女士对于公司只有自己一个测试人员感到很是自豪，这点自信对于测试人员其实是难能可贵的。但是她却有点“闭门造车”，缺乏与其他测试人员以及同行的交流。在我看来，没有看过 Android 开源文档的测试和没有接触过 Android 测试之间可以划等号。所以张女士同样没有通过我的面试。

如果你有机会进入一家创业公司并且愿意接受挑战的话，那么这绝对是一个不错的机会。你可能有更大的机会接触到整个项目，你可能会远离办公室政治，远离大公司的那些条条框框，你可能能够全面发展自己的能力，而不仅限于测试。

我相信肯定有朋友觉得疑惑：上面说的都是进入两种不同企业的优势，但那么多的“可能”。我们怎么才能增加“确定”的概率呢？这需要正确地审视这家企业，审视自己。你需要在面试结束之后让自己清晰地了解以下几个问题：

- 这家企业做什么产品，我是否有兴趣。
- 这家企业目前有多少测试人员，分别是负责什么工作的，目前整体的测试深度大概达到什么程度。
- 自己进入这家企业具体要做什么测试：是功能测试，还是压力测试，还是各种都会接触。企业管理层对于测试人员的职业规划以及将来的发展是怎么考虑的。

可能有的朋友会说，我去面试，不见得所有的企业都会告诉我以上这些问题的答案。的确，作者也遇见过这样的企业。但是，试问如果在面试的时候企业都不能回答你这些问题，这样的公司你敢去吗？你真的愿意去吗？难道只是为了眼前许诺的那些薪资？希望大家好好思考一下。

1.8 测试外包

如果你想找一份测试工作的话，那么对外包这个词绝对不会陌生。在业界，似乎测试和外包是双生儿，有测试的地方必有外包，而有外包的地方必有测试。国内外有名的外包公司都是以输出测试人才为主要业务的。我个人并不排斥外包这种形式，身边很多能力强的前辈都是外包出身。但是需要注意以下几点：

- 要前往好的项目组工作。
- 如公司内部没有很好的流程和技术积累，最好要外派性质。
- 一般外包工作不要超过 2 年，但也不是绝对的，视具体情况而定。
- 看清楚即将签的合同。

从行业普遍的情况来讲，外包性质的工作都是去条件相对较好的国内外大企业，并且入职要求比正式员工低很多，对于刚加入测试行业的朋友来讲是一个不错的选择。但是如果不仔细挑选的话，很容易成为廉价劳动力，工作也没有归属感。所以当你决定从事外包测试工作的时候，一定要慎重。

1.9 小　结

本章介绍了我个人对面试的一些看法。也许现在看来已经有些老套了——当大家看到这本书的时候，移动互联网的发展又到了一个新的高度，无论测试还是技术都日新月异。对面试的革新也很快，要想抓住好的机会就得提前准备——提升自己的能力以及保持良好的心态。我相信你一定会进入想去的企业并最终获取成功！

读书笔记

第 2 章　病态的现状

随着移动互联网的快速发展，跻身移动互联网的企业也越来越多，移动应用产品更是百家争鸣、百花齐放。移动应用已融入并悄悄地改变着我们的生活：我们开始随处签到，随处拍照，随处发微博，上下班时在地铁上看到越来越多的人拿着手机玩游戏和看电影。有需求就会有供给，于是大量的从业人员或是应届生加入了这波移动互联网热潮，而各种风投和天使资金的流入也让移动互联网行业像打了激素一般飞速成长。但是任何行业或者领域如若发展不均衡，都会长成畸形。横观整个互联网行业，有人天天数钱，有人天天输钱；有人如沐春风，有人苦不堪言。病态，这就是现状。

2.1 全民挖金

早在 2009 年初，国内只有极少数的创业公司想到了进入移动互联网做基于 Android、iOS 平台的应用产品。在当时，更多的应用是基于 Symbian 和 j2me 的平台的。到了 2010 年，随着国内使用智能机的用户比例直线上升，各个互联网巨头开始行动起来，纷纷招兵买马，在移动客户端上扩展自己的业务。随着“水果忍者”、Testin、Path 靠着一个游戏或者仅凭一个新业务的想法就拿到了很多投资，越来越多的巨头和敏感人士觉得新一轮的淘金浪潮又要开始了。



在众多的故事中，Draw Something 的一夜暴富更是激励了大众，新闻宣传扑天盖地。该企业一直在做游戏，连续做了多款但收入微薄，濒临破产之际愤力一博，作出了一款移动互联网产品，不但拯救了全公司，甚至还一跃成为“当下红人”。

但是随着时间的推移，更多的人发现貌似第一桶金来的也不那么容易了。这桶金的来源一般有以下几类（并非全部）：

1. 企业自己出资

作为传统互联网的霸主，谨慎是他们的品质。他们不会轻易涉水移动互联网，但也不会轻易放弃移动互联网。所以在早期，这些巨头都拿出了一部分钱投在了移动客户端的项目上。当然，对开发移动客户端的团队来说这自然就是第一桶金。不过在这次浪潮中，很多团队发现这桶金是连响都没听到就打水漂了，更有甚者还影响到了企业原有的用户群。的确，早期移动互联网的试水鲜有成绩。

2. 市场标价

这里所谓市场就是指 Google Play（之前称为 Google Market）和 Apple Store。普通产品的标价一般最低是$0.99，而游戏的标价普遍偏高。比如最红火的“极品飞车”、“最终幻想”系列在高清移植到智能机上之后，高调标出了几十美元的价格。

在早期，企业或者个人靠市场标价还能有一笔不错的收入，但是随着时间的推移，免费应用越来越多。而 iOS 系统完美越狱的出现瞬间导致了市场上绝大部分的收费应用无钱可取，众多企业甚至关门大吉。直至今天，只有极少数的应用能够坚持收费并且继续发展。这里值得一提的是，随着移动互联网的发展，个人开发者越来越多地加入，但由于苹果高昂的手续费，黑卡盛行，再加上国内版权意识薄弱等因素的影响，可以说个人开发者想通过市场标价获得第一桶金已无可能。

3. 天使&风险投资

这也是很多移动互联网企业最常见的一种资金来源。从 2009 年开始，不断地有传闻说，只要一个小小的想法就可以融资几百万甚至几千万！这让越来越多的人似乎发现新的致富之路，原来不买彩票也有机会变成百万富翁。创业热潮就这样开始了，他们三五成群开始了天马行空的想法，为的是拿到 A 轮投资。这些人中大多是很有想法，抱着远大理想的年轻人。他们以为只要有技术和好的想法就能够成功，殊不知创业远远比他们想像的艰难，而融资更是为别人做了嫁衣。

4. 产品定制

这是我认为最稳妥的一种挖金方法，根据大公司的要求制作一些特定的应用，最好是智能机默认的应用。比如短信、输入法、联系人等强需求的软件。采用这种方式的前提是必须有能力足够好的销售人员去说服大公司，也需要有能力强的技术人员在客户要求的时间内完成产品开发。任何一个错误都有可能让一家创业公司失去生存的机会。

5. 免费应用内部收费（in-app purchase）

事实上有个别企业因为此类收费模式赚到了很多钱。不知从什么时候开始，我们玩的一些免费游戏里除了成就系统之外，不知不觉就出现了“商店”这样一个概念，如

果我们想要获得更高的游戏分数那么就需要购买收费道具。其中最杰出的必属于 Clash Of Clans，该游戏让黑卡在淘宝上再次疯狂销售。接着，其他免费应用也出现了收费项目，比如微博的会员、拍照软件的收费滤镜等等。这类隐形收费的目的就是为了避免广大用户不会因为看到软件收费而连被下载使用的机会都没有了。但是由于 Android 太过开放、iOS 系统越狱，大部分内部收费应用也是昙花一现。

6. 广告

广告是一个在任何行业、任何模式下都不会消失的硬渠道。在移动互联网，广告更是猖獗。每次打开应用的等待时间、游戏娱乐、查看微博，都会出现它们的身影，有些甚至还有不知不觉帮助用户自动下载并安装广告软件。广告是一把双刃剑，广告的费用一直是非常昂贵的，投入得恰当可以带来更多的下载量和更大的影响力，反之则可能给自己带来毁灭性的打击。

从 2011 年开始，各个天使和风险投资的数量和数额都明显地呈下降趋势。即便如此，各行各业的人还是继续在移动互联网淘金，正式开启了全民淘金的时代。

2.2 别人的嫁衣

许多年轻的创业者揣着自己琢磨了许久的想法去和投资人谈合作，拿到投资方的钱以为离自己梦想又近了一步，但实际上可能是离投资人的梦想又近了一步。

很多在创业公司工作的人都会感觉到流程混乱无序，这是因为创业者真的很忙。他们要忙着公司法律相关事务，忙着不停地抢注专利来保护自己的企业，还要忙着不停地忽悠投资人以求继续发展。我们常常说创业的人都需要有一技之长，看待产品的视角要远于常人，但是又经常感觉他们的想法一天一个样，有时候甚至背离最初的想法，这大概与投资人的加入有关。投资人觉得自己出了钱，那么这家公司的产品及其发展方向就得关心。随着时间的推移，想法还是创业者的想法，但是产品已经再不是当初创业者心中所想的产品了，而是变成了投资人手中如何快速赚钱的工具。

最近很多人和我提到自己是多么仰慕微信和张小龙。我个人对张小龙不做评价，但是我想将 talkBox、米聊、飞信 3 个软件放在一起讨论。iOS 版的微信 1.0 诞生于 2011 年 1 月 21 号，我们来看看这 3 款应用，不知道对于已经被微信洗脑的广大群众看来是什么感觉。

2.2.1　TalkBox

我使用的第一个语音社交类产品就是 TalkBox（也许关注此类产品的朋友还会想到 kik）。Talkbox 最初是由 6 人组成的创业团队，针对开车不方便接听电话的人以及盲人开发的全球第一款语音 IM 应用。于 2011 年 1 月上线，仅仅 3 天的时间，在 App Store 的下载量就达到了 100 万。在腾讯推出微信之后，TalkBox 的占有量直线下降，最终 TalkBox 决定扭头转向海外市场。



2.2.2　米聊

在使用 TalkBox 不久我便装上了米聊，纯粹是为了体验新产品。米聊比微信早一个月推出，其最初的用户增长量和媒体关注度远远超越微信。2011 年 5 月 10 日，微

信 2.0 版本发布。在此之前，米聊已先行一步推出语音对讲功能，用户反响强烈；TalkBox 英文版也在国内拥有一批“高端粉丝”。之后，微信迅速推出了“摇一摇”和“朋友圈”的功能，从而奠定了胜利的基础。



2.2.3 飞信

最后我们来说说飞信。我使用飞信主要是两大原因：其一是朋友们都在用，其二是可以用自己的移动号登陆并免费给其他号码发短信。

飞信 Fetion 是中国移动于 2007 年 5 月推出的“综合通信服务”，即融合语音（IVR）、GPRS、短信等多种通信方式，覆盖 3 种不同形态的客户通信需求，实现互联网和移动网间的无缝通信服务。飞信也许是移动做过的最大的、也是唯一的一个社交应用了。

不过说句实话，我个人觉得飞信的没落和微信关系不大，属于“自作孽”。



大家是不是对这段缠绵纠结的故事有一个大概的了解了？所以说，如果在那个时间段，有一家创业公司做一个产品叫做“微信”，那么现在的这个“微信”和上面那 3 者的下场也差不多。在我看来，最大的区别在于微信有着自己的产品定位和策划，并很好的利用了腾讯的用户群。现在米聊等产品已经是一个“四不像”产品了，也许一部分已经是投资人的了。

如果在读这本书的你正在为投资人做嫁衣的话，也不要气馁。因为在移动互联网实在有太多的人都挣扎着没有放弃自己的梦想，我衷心地为你们加油。

2.3　超越光速的迭代

仅仅有想法是没用的，接下来就需要实现。就我目前的了解，移动互联网应用的平均迭代速度远远超过了传统互联网的任何一个产品，相比以前互联网的项目迭代速度真可以称之为“光速”。在移动互联网，一般项目的开发周期在 1～2 个月，而项目的版本甚至一天就会迭代好几次。同传统的软件开发相比，这种巨大的差距来自于移动互联网的特殊性。

1. 创意高于一切

假设有一个应用项目，需求在 2 个月内完成 20 个功能。很有可能，在某一天熬夜写代码之后吃早餐刷微博的时候，发现另外一个有着相同创意的应用已经在市场发布了，并且在微博上已经有数万转发。那么在移动互联网会发生这样的举动：仅仅完成创意的功能，其余的可以全部不做，尽量赶在另外一个应用发布前发布，这就是现状。用户不会关心创意是谁的，先入为主很重要，在他们看来其余的都是抄袭第一个应用的。

2. 成本低

在移动互联网，一个应用制作到发布的成本都很低，一两个开发人员就能够在短时间内做一个可发布的应用。对于企业来讲也是这样，要发布一个应用到网络市场是非常简单的事情。即使用户在使用应用中出现了任何严重的问题，企业也能够在很短的时间内快速修复问题再发布（当然要排除一些金融、医疗等行业的应用，由于行业关系，它们依然存在高风险性）。对用户而言，下载一个应用并安装使用的成本更低，毕竟人类是一种喜新厌旧的生物。

3. 需求不明确

需求一般有 3 个来源。产品经理、客户和用户。

- 用户永远不会满足于自己已经使用的应用，他们一直期盼软件更新以满足更多的需求；
- 客户一般是周期性地提出需求，虽然有时候会不明确，但不是最终导致快速迭代的原因；
- 剩下的就只有产品经理。团队获取需求的来源其实都是通过产品经理，而大部分需求不明确的原因也是因为产品经理错误地理解了用户和客户的需求。

谁的错？当然，也不能说都是产品经理的错，往往团队本身就缺乏一种很好的沟通流程来明确需求。此外，还有一部分原因是一些产品经理每天都有天马行空的想法。

在这样快速的迭代周期中，开发人员满足需求已是难事，事实上，能留给测试人员的时间是少之又少。目前，一般的 app 开发都会采取很多公司老板和员工自创的“敏

捷模式”，其本质就是压榨程序人员：开发人员为新功能疲于奔命，测试人员既要测试新功能，又要做老功能的回归测试，这样一来必然纰漏百出。所以，更需要测试策略和高效的测试方法来支撑，相关内容会在之后几章详细介绍。

2.4　可怜的测试团队

在 IT 行业内，有一个问题一直被讨论着：“在一家企业中，开发人员和测试人员的合理比例是多少？”。每当听到此类问题，总有人会说“微软公司的开发人员和测试人员比例是 10：1”或者“将来没有测试人员了”。随着移动互联网产品越来越火，敏捷测试也跟着流行了起来。敏捷测试提倡“全民测试”，也就是企业内部无论什么职务的职工都是测试人员。那么，测试人员占全公司的比例应该是多少呢？

问大家一个问题，如果你要做一个移动应用，在如此快的迭代周期，处于如此激烈的竞争环境下，是先选择招聘开发人员还是测试人员呢？答案可想而知。无论是大公司还是创业公司，团队人员或是技术都是第一位的。那么，在一个团队中，测试人员的比例应该是多少，我想大家心里已经有答案了。不过，我还是提供一个实际的信息，在初创团队中很少看到测试工程师。创业者会告诉你，产品都还没有成型，没有必要去招测试工程师。在大公司中，往往几个产品多个平台并行开发，比如曾经有一家大型搜索公司，两个产品四个平台，功能测试和自动化测试工程师加在一起，也不过 10 人。

测试团队不但人数少，还被极大地压缩测试时间，这在移动互联网行业中已经见怪不怪了。也许有人会说很多团队就是要小而精的。那么接下来就说说团队质量如何。对于创业公司来说，虽说已拿到了投资，但毕竟都要用在刀刃上。团队中往往有几个资深的开发人员以保证产品的正常上线，测试人员则多数是由应届生和实习生组成的。大企业会组建一个测试团队，团队中的测试经理一般由企业内部转岗人员担任，其他成员则由应届生或是测试外包工程师组成。由此可以看出，移动互联网行业中大部分的测试团队不但人数比例小，而且平均测试技术也不强。

2.5 敏捷中毒

无论项目团队是否实施敏捷，整个行业的节奏正逼迫着所有的团队在向敏捷靠拢。2001 年提出的“敏捷宣言”是敏捷历史上的一个里程碑。敏捷宣言如下：

个体和互动 高于 流程和工具

工作的软件 高于 详尽的文档

客户合作 高于 合同谈判

响应变化 高于 遵循计划

也就是说，尽管右项有其价值，

我们更重视左项的价值。

从敏捷宣言中不难看出，它并不否认传统项目开发中的很多内容，但更强调沟通，强调随机应变。由于现在大部分的企业和员工并不了解敏捷开发的实质，所以现状就演变成了：

只要个体和互动 不要 流程和工具

只要软件在工作 不要 详尽的文档

只要响应变化 不要 遵循计划

很多测试工程师认为敏捷就是不再需要测试计划、测试用例，只要自由测试就可以了。恰巧在同一时期又出现一个新概念——“探索性测试”，这让测试工程师们更坚定了随机自由测试的谬论。敏捷和探索性测试的方法、流程不在本书探讨范围之内，感兴趣的读者可以自己了解。

2.6　无力的测试

自 Google 公司的 Android 系统面世以来，机型层出不穷，各种各样的 Rom 也是铺天盖地。在短短 4 年的时间内 Android 从最早的 1.0 版升级到了现在的 4.2 版，升级速度实属罕见。为保证应用可运行在不同版本的 Android 系统下，就必须全面测试应用的兼容性，这样就大大增加了测试工程师的碎片化测试（直到我写这本书的今天，Android 2.3 依然拥有着一部分受众），这对于开发和测试都是极大的挑战。

相信所有 Android 的开发工程师都不会忘记，为了让应用能够在 Android 各个版本的智能机上获得更好的体验而一遍遍地进行适配的痛苦。而此时测试工程师们面对的是一个基本上无解的问题——Android 智能机的适配测试该怎么做？在这个问题面前，任何精湛的测试技术都变得苍白无力。

相信很多做过 Android 应用测试的人都会有这样的经历：在公司内部做的测试已经自信满满地结束了，但是一经发布，到用户使用的时候，各种明显的缺陷报错瞬间可以淹没企业的通讯系统。我相信每个经历过的人都曾经为这样的情况而倍感无力。



Android 4.0 以前各代版本 Loge

此时有两个看似很不错的解决方案放在眼前：一是使用 Android 模拟器，二是使用 Testin 或 MTC 这样的兼容性测试平台。不过我可以很负责地告诉大家，截至到本书成书之时，这两者都是不可靠的。

Android 模拟器的确可以模拟各种设备的分辨率以及各个版本的 Android 系统，但是，由于模拟器是 Java 虚拟机构成的，在任何配置的 PC 上其性能表现都很慢。就我两年测试国内外各个智能机的经验来讲，模拟器中的应用显示以及功能与实体机还是相差很远的，更不用说像小米或索尼爱立信这类定制的智能机。

Testin 是一家我很看好的公司，在 2012 年 12 月我还有幸去该公司，学习他们如何做大量的智能机应用的自动化测试。但我前面说不可靠的原因基于两点：

其一，无论 Testin 还是 MTC，目前的兼容性测试服务做的还不够成熟。使用过的人都知道，现在移动互联网兼容性测试平台已经能够很好地支持在各种智能机上面进行安装、卸载、打开、MonkeyTest 等测试，但是仅仅这些测试并不能满足日常的测试要求。

其二，这类测试毕竟是借助第三方平台进行的测试，如果上司或客户让我给出一份产品的兼容性测试报告，我是无论如何都不可能将第三方平台的测试结果给他们看的。毕竟作为测试工程师，我只相信真正测试过的以及我所看到的，这类测试报告只能当作参考。



提示： 那么朋友们肯定要问：“我们测试工程师该怎么办？”对该问题的解答将在后面几章中介绍。

苹果的 iOS 在兼容性方面比 Android 出色得多，但是同样使测试工程师的测试感到非常的无力。兼容性好不等于不用测试，且苹果的 iphone、ipad 价格一直不低，一般企业管理层认为各有一台就足够了，这给苹果应用的兼容性测试带来了极大的困难。随着苹果 iOS 系统遭破解（也就是所谓的越狱），出现了很多系统插件，这类插件在某些情况下会非常大地破坏 iOS 原本的生态平衡，也使得原来能正常运行的软件变得不稳定，自此越狱系统也被纳入了不得不测试的范围。此外，苹果的 app store 审核机制对测试工程师来讲更是雪上加霜，苹果应用的审核一般要一周左右的时间（节假日不包括），意味着产品发布之后如果有严重的问题，不能马上用一个新的版本替换，只能先从市场上下架。所以 iOS 应用的测试工程师压力相比 Android 的只会更大。

那么移动互联网的测试工程师到底应该怎么做测试呢？如何改变这样的现状呢？本书的各个章节只是给出了一些引导方向，更多的还是需要大家根据公司、项目、人员等具体情况具体分析。

2.7　浮躁的测试工程师们

本节仅仅写给测试业界里那些浮躁的测试工程师们，切勿对号入座。测试业界中除了一些技术牛人和一些会忽悠的“大佬”以外，剩下的工程师们大多挣扎在各个企业中。他们不仅要和开发做斗争，还需要和生活、和薪资做斗争。从长期来看，这些人大部分就是在浪费时间，不学无术，但他们自己却还觉得社会很不公平，各种委屈。我列出一些浮躁行为，大家切勿对号入座：

2.7.1　社交中毒

我自己在刚学习测试的时候加过非常多的群，原以为能够在有问题的时候找到几

个“高手”来解决问题。直到加了之后，我才发现几乎所有的群都是整天在聊天，无任何技术讨论话题。群中关心的问题无非是：

- 薪资。别人拿多少钱，自己拿多少钱，然后开始讨论怎样才能够拿到更多的钱。
- 抱怨。抱怨团队，抱怨流程，抱怨老板。抱怨一切可以抱怨的事情。
- 公司。大家开始晒各自的公司，大公司的人就会受到很多人的敬仰。
- 求职。不停地销售自己。

QQ 的确是一个好软件，在中国长期以来积累了大量用户并屹立不倒。但它就和毒品一样让人容易上瘾，在不知不觉中浪费了时间，也将全部整块的时间碎片化。现任豆瓣技术总监的段念曾经告诉我，不要企图在 QQ 群里得到很多核心的技术，那里更多的只是在浪费时间。我现在也越来越深地体会到这点。

我只想问那些在 QQ 里讨论以上几点的测试工程师们，你们与其在群里聊天，不如将这些时间用来学习，你们觉得这个解决方案怎么样?

2.7.2 失去自理能力

在我长期 QQ 隐身在线期间，看到很多测试工程师作为一个健全的，甚至是一个技术人员，却完全不会自理。比如以下几个问题经常会连续的出现：

- 我想学习 MonkeyRunner，请问 MonkeyRunner 能做点什么?
- MonkeyRunner 环境已经配置好了，请问怎么写？有没有什么例子?
- 请问 MonkeyRunner 除了模拟用户操作还能做什么?
- 有没有什么文档?
- ……

这些问题不仅仅出现在移动互联网，只要这些测试工程师们不改变自己学习的态

度和习惯，那么在任何一个领域、任何一个技术上他都会出现同样的问题。他们面对一个对他们而言未知的事物就完全丧失了自理能力。我能说的是，Android 或者别的所有技术都能够在其开源文档或者 WIKI 找到很详细的描述，请自行学习。

2.7.3　不能正确认识自己的价值

我在新浪微博上曾经发过以这样一段对话：

A：“你好，你这边招聘测试工程师吗？薪资大概多少？”

我：“你好，具体看应聘者的能力而定。”

A：“我以前就是做 Android 的，有 5 年工作经验，很符合你们的要求。”

A：“还在不在？我是本科毕业，做 5 年大概能拿多少钱？”

A：“这个是我的简历，5 年你差不多给个价就行了。”

我：“......”

很多朋友肯定会觉得上面这个测试工程师怎么那么傻，自己肯定不是他这样的。不过从事实来看，测试业界的确还有不少这样的“工程师”。他们有几个特点：

- 对别人的薪资很感兴趣。
- 不务实，不知道如何自己解决问题。
- 只从薪资、学历、工作年限来描述一个工程师。

一名测试工程师的价值到底是如何考核的？绝对不能仅凭学历、工作年限、薪资等来评价，关于这点在第 1 章已经详细叙述了。同样是测试工程师，有的人一边工作一边总结，同时和业界的其他工程师沟通交流，每天都有进步。而有的人仅仅是在劳作，每天做着重复的业务，工作十年二十年和工作头一年没有区别。我们需要正确地去评价和看待自己。

2.8 测试沙龙和培训

相信测试业界的工程师们都有这样一个感觉，近几年来测试业界大大小小的沙龙越来越多。在国内，每人费用四千多的沙龙也有。大部分的沙龙做分享的人是学术派、理论派的所谓的“老师”，他们可能以前做过测试，只不过时间早了一点，但是长期脱离一线之后，他们剩下的其实也只有一些理念和经验罢了。有一部分前辈的确是想将测试做好，是想帮助更多的人。但另一部分人就纯粹是为了自己的利益，为了赚到更多的名声和金钱而在各个大小沙龙“扯淡”，很多没有分辨能力的人受迷惑也正中了他们的下怀。

那么，沙龙真的对我们没有帮助了吗？我们去参加一个沙龙活动，首先要想清楚自己去的目的：可能是为了解决某个具体的问题，可能纯粹是想了解一下别人使用的技术和流程，也可能是为了得到某些启发。无论是哪种，只要想清楚之后再参加沙龙，才会真正的有所收获。

我个人最喜欢的沙龙形式是互动的形式。站在台上的人不一定比台下的人都厉害，大家是平等的，可以非常自由地针对某个问题进行讨论。与其一味地灌输理念和方法，不如一起交流，去切实解决大家心中的疑问。沙龙的内容必须与时俱进，很多业界的前辈一直在说一些过时的理念和概念，而一说就是好几年。我个人认为这些人可以讲，可以分享，但适合以个人免费沙龙的形式开展，而不适合在大型的收费沙龙上去“扯淡”。

与沙龙在同一时间流行起来的是测试培训。除了 51testing、达内、北大青鸟等大家耳熟能详的一些机构之外，各个小公司甚至个人也都展开了测试培训。培训的形式从以前的线下培训变成了网络培训。收费一般都在 5000～20000 元不等。很多想做测试的人碍于没有基础想去参加培训。我不会评论哪些机构好或者不好，但我希望去培训的朋友要了解清楚以下几点：

- 自己是否对培训的业务和技术领域感兴趣；
- 培训课程是否与时俱进；

- 培训的老师是否值得信任；
- 自己是否清楚培训结束之后能够真正的得到什么。

不要被老师的头衔和包装过的课程所蒙骗，仔细地了解清楚之后再去参加培训也不迟。

2.9　学会自我尊重

很多测试工程师在工作中往往会觉得不受重视或者被别的同事看不起。其实外因有两个：一个是国内大部分企业根本还不知道怎么正确地考评测试工程师的 KPI（Key Performance Indicator，关键绩效指标），另一个是国内大部分非测试行业的从业人员也不清楚测试工程师的具体工作是什么。而内因只有一个，就是这些测试工程师自己都不知道要做什么，没有找到自己在企业中存在的价值。请这些测试工程师们扪心自问一下，你们自己都看不起自己，为何还要抱怨别人看不起自己呢?

测试这个行业和其他行业有一个本质上的不同，这里暂时按下不提，先拿基于互联网基础的其他行业来比较。

- 开发工程师：在某个周期（schedule）内完成某些特定的功能（Feature），是将产品从理念转换到可视化生产，对外是一个从零趋向于正区间的概念。
- 销售人员：一般是本金再加上提成。简单来讲，是为公司争取了多少合作方和合作项目。对外是一个从零趋向于正区间的概念。
- 市场运营：在某个活动中利用自己的人脉资源和各种渠道进行推广，最后实际看到活动参与的人数以及影响效果。对外是一个从零趋向于正区间的概念。
- 项目、产品经理：在一个周期内管理，协调完成一个产品。同样的，对外是一个从零趋向于正区间的概念。

从以上的描述很容易看出，国内大部分以结果为导向的企业很容易借此标准去评

定这些员工的 KPI。但是测试工程师不同，测试工程师对外的概念如下：

- 测试工程师：保证发布的产品没有缺陷。

是不是很简单？我相信在很多人的眼里就是这样的。但是事实来讲，没有一个产品是没有缺陷的，当产品通过测试还发现有缺陷的时候，自然地，很多不了解测试的人就将矛头指向测试工程师。所以测试工程师对外是一个从零趋向于负区间的概念。这也正是很多测试工程师觉得自己不受重视，工作苦不堪言的根本原因。

前不久出版的《乔布斯自传》是一本很不错的书，老乔在最早要建苹果公司的时候和现在广大的创业公司一样，他也会去找合伙人，去推销自己的产品。“你想和我一起改变世界吗？”，相信很多人都期望老乔能够和自己说这句话。先不论人家企业是怎么成功的，单是创始人对自己以及自己做的产品充满着任何人都不能相比的信心，就知道他们会成功。因为他们就是要让全世界都知道他们的企业，知道他们的企业改变了人类的生活，而这个过程就是一个改变用户，让更多的用户了解产品、接受产品、喜欢产品的过程。

测试工程师在国内的角色从某些角度来看和这些企业的创始人一样，“测试”就是我们的产品，而我们要做的第一步就是学会自我尊重，对自己和测试本身都要充满信心，然后我们才能够去感染身边的人，让更多的人了解测试，正确地看待测试，喜欢测试这份职业。

2.10 小　结

本章从各种角度阐述了目前移动互联网测试行业的现状。测试行业在中国的发展历史并不长，但是目前规模已经非常庞大，由此可见是一个很有前途的行业。在这样一个环境中，我们需要出淤泥而不染，需要随时告诫自己入行的初衷是什么，需要定期去审视自己是否有所提升，更需要定期与同行交流，避免自己闭门造车。在此，我呼吁测试同行们，不要害怕，不要自卑，让我们放手去选择我们喜欢的团队、产品和企业吧！让我们一起改变中国的测试现状，让国内的测试工程师走出国门！

第 3 章　用户体验测试

本章是正式讲解移动应用测试主题的第一章，介绍用户体验测试。为什么首先选择这个主题呢？用户体验在移动互联网中是很重要的测试点，但同时也是不为广大测试工程师所重视，甚至认为不在测试人员职责之内的一项工作。其实，在移动互联网激烈的竞争中，产品的用户体验甚至比其功能更重要，换句话说，功能的完善最终都是为了更好的用户体验。实际上，整个项目团队中每个成员都应该对产品做用户体验测试。不同职位的人关注点不同，测试的切入点就不同。这点 Google 做得非常好，拿搜索引擎来说，每个版本都有内部版本，每个员工都要使用，也会提出合理的体验建议。那么，对于专业的软件测试工程来讲，用户体验测试到底是什么呢？需要测试哪些点？

3.1　移动互联网与传统互联网体验上的区别

在移动互联网发展的初期，大部分产品的界面、交互都是根据互联网时期获得的经验来设计的。随着移动互联网的发展，大家发现传统互联网产品的用户体验在移动平台上表现得糟糕透顶，因此需要更加适合移动互联网的设计。

作为一名移动互联网应用测试工程师，需要对产品有着很好的使用经验，这份经验将是引导很多测试活动的关键。很多想从事移动互联网行业的朋友也经常会问两者的区别到底是什么，用户体验是一个切入方向，能够决定产品将来的方向。

3.1.1　区别一——用户关注点

用户通过电脑在传统互联网上浏览信息时，由于电脑屏幕很大，可以同时显示更多的信息。所以，我们能够看到很多网页打开之后密密麻麻的信息。



这种主页的设计类型几乎在各个门户网站的首页都能见到。那么，在移动互联网中能看到这类设计吗？我相信，既使有的话，这家公司也已经关门大吉了。对于用户来讲，移动互联网一个巨大的改变就是他们要将视线从几十英寸的电脑屏幕转移到一个只有几英寸的小屏幕上，能够显示的信息就变得有限和珍贵。这就更需要将有价值的信息放大，放在应用中显眼的位置，否则必将被用户淘汰，最后只能是删除这个应用。

3.1.2　区别二——场合复杂化

对于传统互联网，用户会在家里、网吧、学校等地方使用电脑，至少我相信很少有人在上厕所或者地铁上使用电脑吧（工作需要除外）。但是对于移动互联网，更多的用户会在挤公交、上厕所甚至运动的时候使用智能机。因此我们在测试应用的时候就必须要考虑到各种情况，比如网络不稳定、智能机电量不足以及在各种场合下用户的操作习惯等。

3.1.3　区别三——时间碎片化

用户使用移动应用的时间大多都是碎片化的，看微博、看新闻，玩小游戏等等。那么应用在设计时就需要考虑到要符合用户碎片化时间中的使用习惯，能够在碎片化的时间内阐述表明自己应用的独特之处，千万不要奢望用户在连续使用半小时之后才理解你的用意。用户只会告诉你，我们没有那么多时间给你浪费。

3.1.4　区别四——输入困难化

用户在使用电脑的时候，鼠标、键盘都是非常方便的输入设备。但在移动互联网客户端上，应用的输入变得尤为复杂，移动互联网应用的输入一般分为实体键盘和触屏。触屏包括手机和电容笔，而实体键盘则随着触屏技术的发展趋于淘汰。输入方法的改变总会让人一时不适应，尤其是在复杂的交互场景中。例如，现在大部分的应用需要注册登录，用户输入数据的时候如果出现错误，修改其中的某个字符足以让用户心烦。大部

分的智能机只支持触屏软键盘输入，输错的几率还是相当高的。所以，在应用设计和测试过程中输入困难也是个相当重要的问题。

当然，移动互联网和传统互联网体验的区别并非只有这几点，但需要清楚地认识到，界面、场景、交互等并不只是产品经理、交互设计师等要考虑的内容，既然体验会大大影响产品的质量，作为测试工程师更是需要时刻关注。

3.2 Android vs iOS

目前智能机使用量最大的操作系统莫过于 Android 和 iOS 系统。我们先通过以下一系列的对比来了解这两个系统设计上的不同吧。

3.2.1 桌面

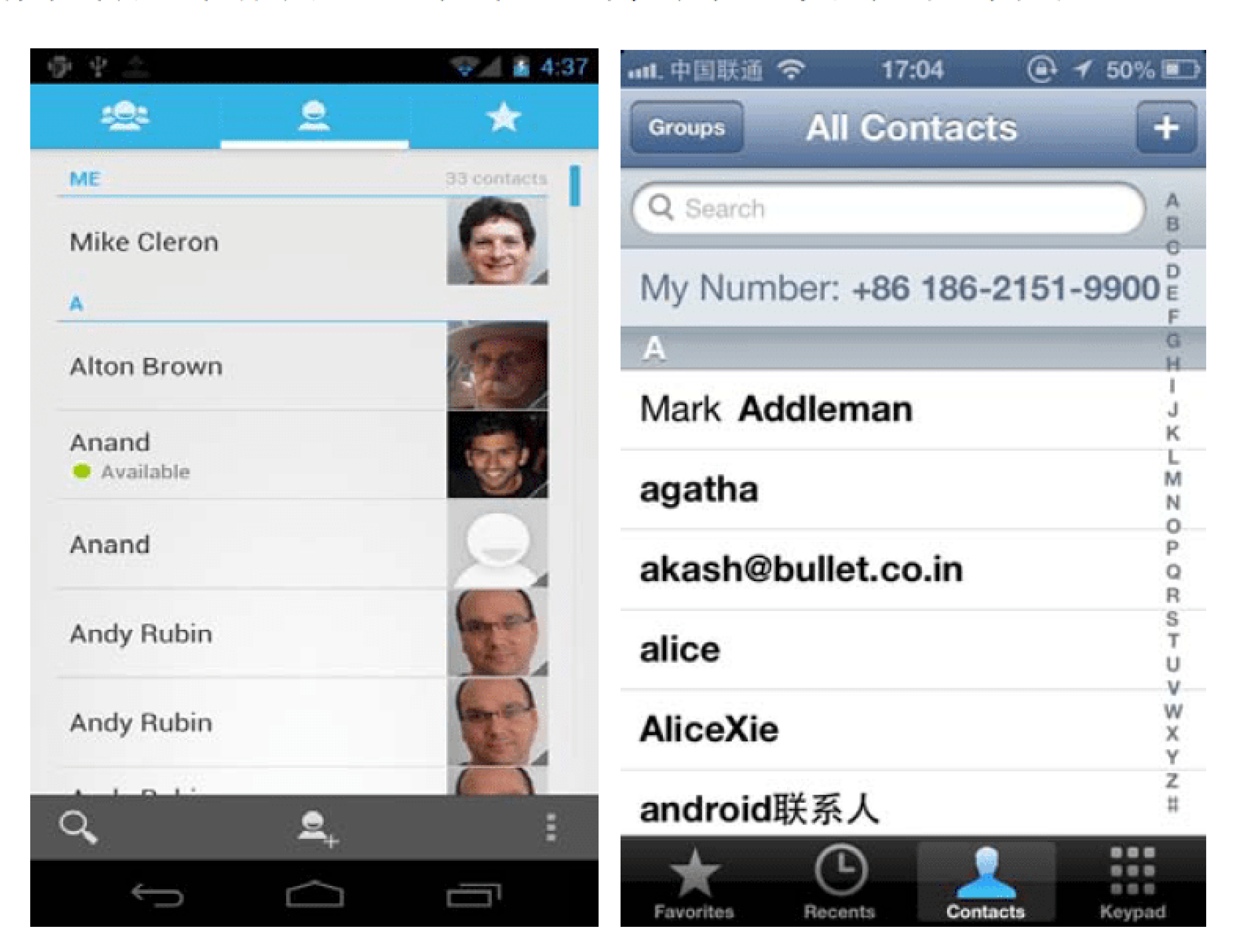


首先是这两个系统的展开桌面（左上图为 Google 的 Android 系统，右上图为 Apple

的 iOS 系统）。老罗（罗永浩，锤子手机创始人）曾经在发布会上面就这两个界面做过对比——我也比较认同这样的观点——无论 Android 还是 iOS 都选择了将应用图标平铺在一个界面上，不同的是 Android 选择了将每个应用图标变小，图标的间隙变大让人感觉整个画面不那么拥挤。iOS 选择保持图标原大小的同时，将图标外轮廓改成圆角正方形，从而让整个画面也显得整齐。两个系统都是以左右滑动的方式进行画面切换。另外，从整体的色彩上看两者也有着鲜明的对比，Android 偏暗而 iOS 偏亮。

3.2.2　联系人

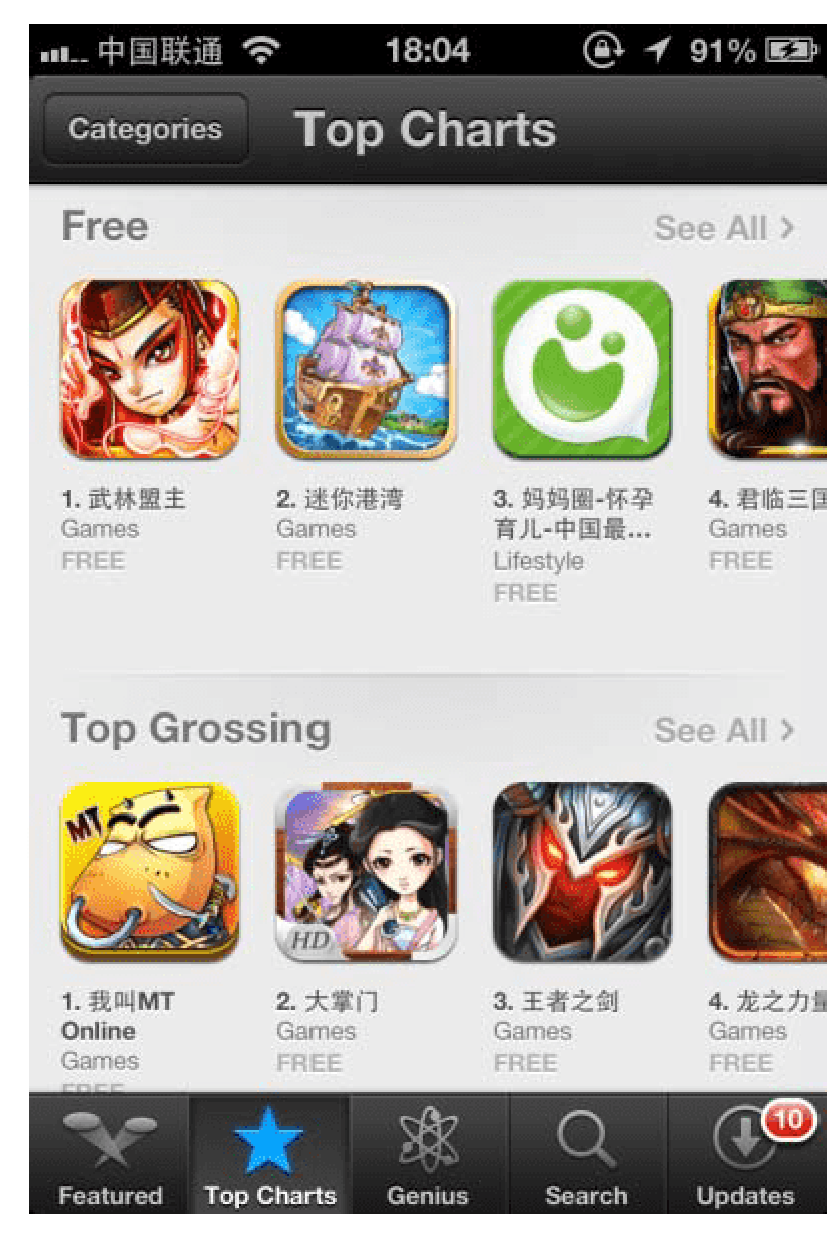
下面两幅图分别是两个系统的联系人（Contacts）应用的界面，风格迥异特点鲜明。Android 和 iOS 都选用了标签栏来进行界面的切换。不同的是 Android 选择小字体并在预览界面显示联系人头像，同时在下方给出联系人应用相关的操作按钮。iOS 依然保持了自己大气的设计，并且为了方便用户快速找到自己的联系人，给出了右侧的快速定位栏；左上角是与联系人应用相关的设置，右上角为添加新联系人，顶部是一个很显眼的搜索栏；底部的标签栏成为 iOS 的经典设计，在其主要的应用中随处可见。



下图从左到右分别是 iOS 系统自带的 Music、GameCenter 和 AppStore。

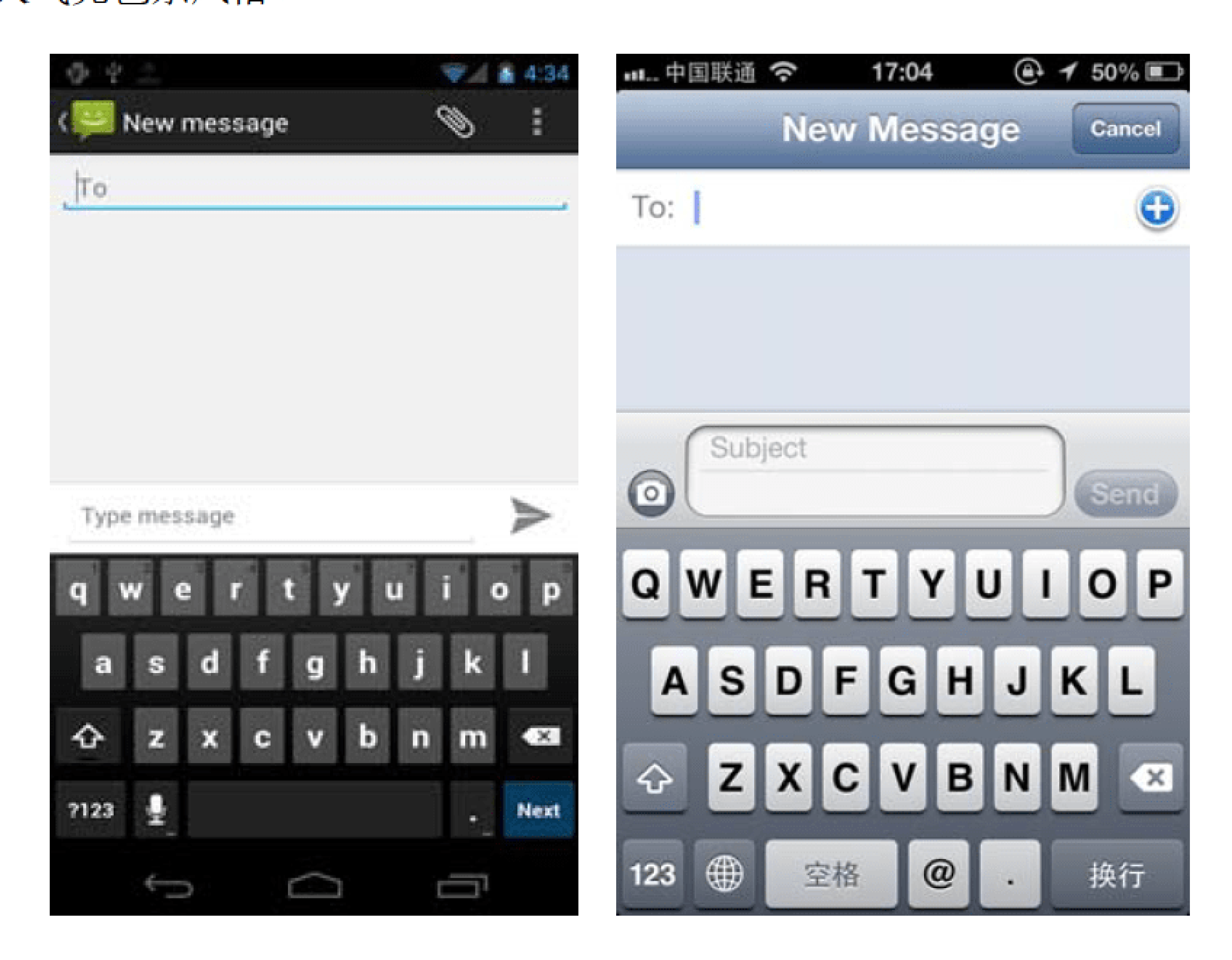






3.2.3　短信

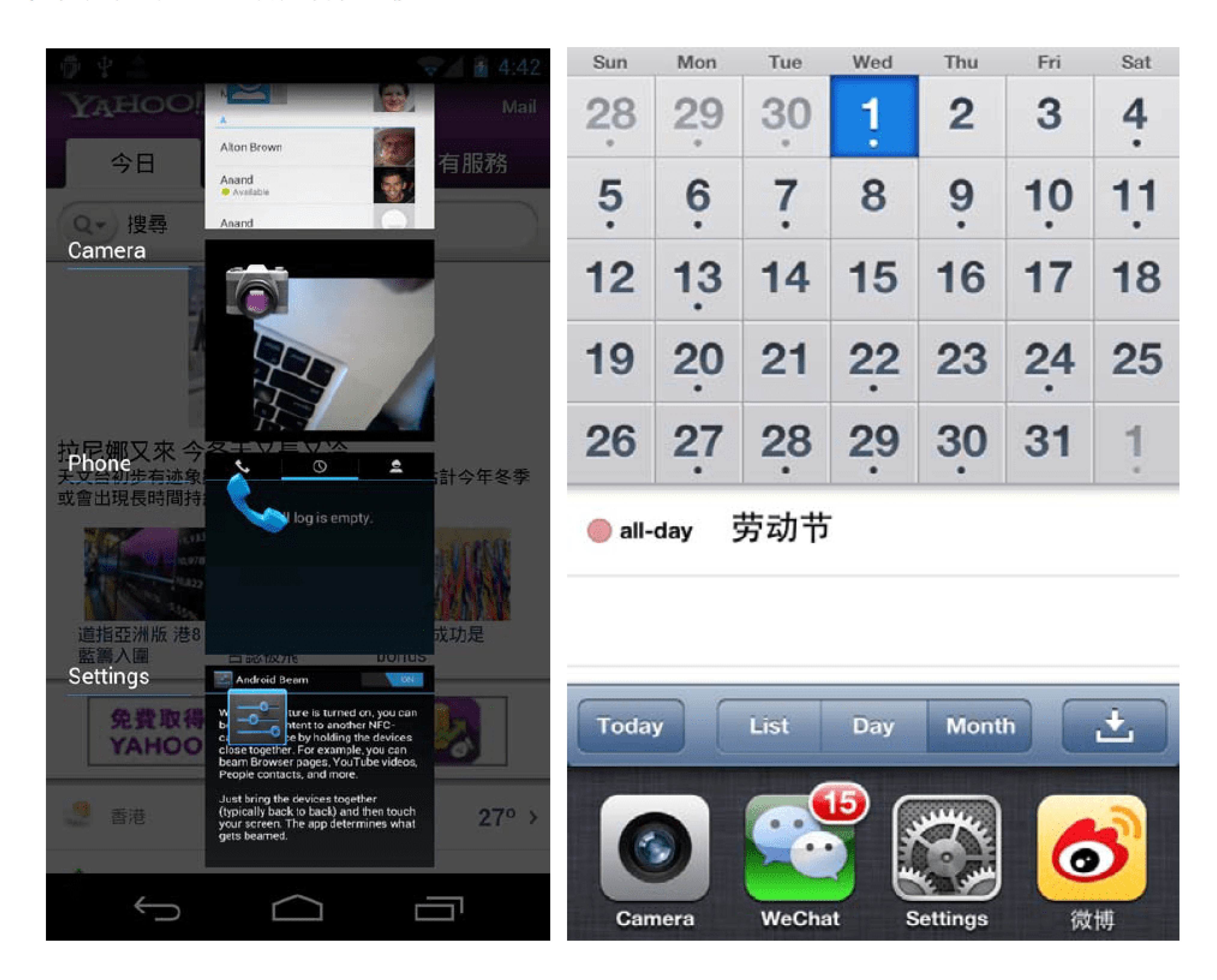
下图分别是两个系统的短信（Message）编辑界面。这个界面相对比较简单，iOS 贯彻了自己“返回在左上方，取消在右上方”的宗旨。值得一提的是，Android 短信界面上方的添加附件按钮及之后更多按钮在 Android4.0 以上版本中也是一个经典设计。另外，从两个系统的默认输入法的设计中也能看到区别，Android 的精简暗色系风格以及 iOS 的大气亮色系风格。



3.2.4　历史应用

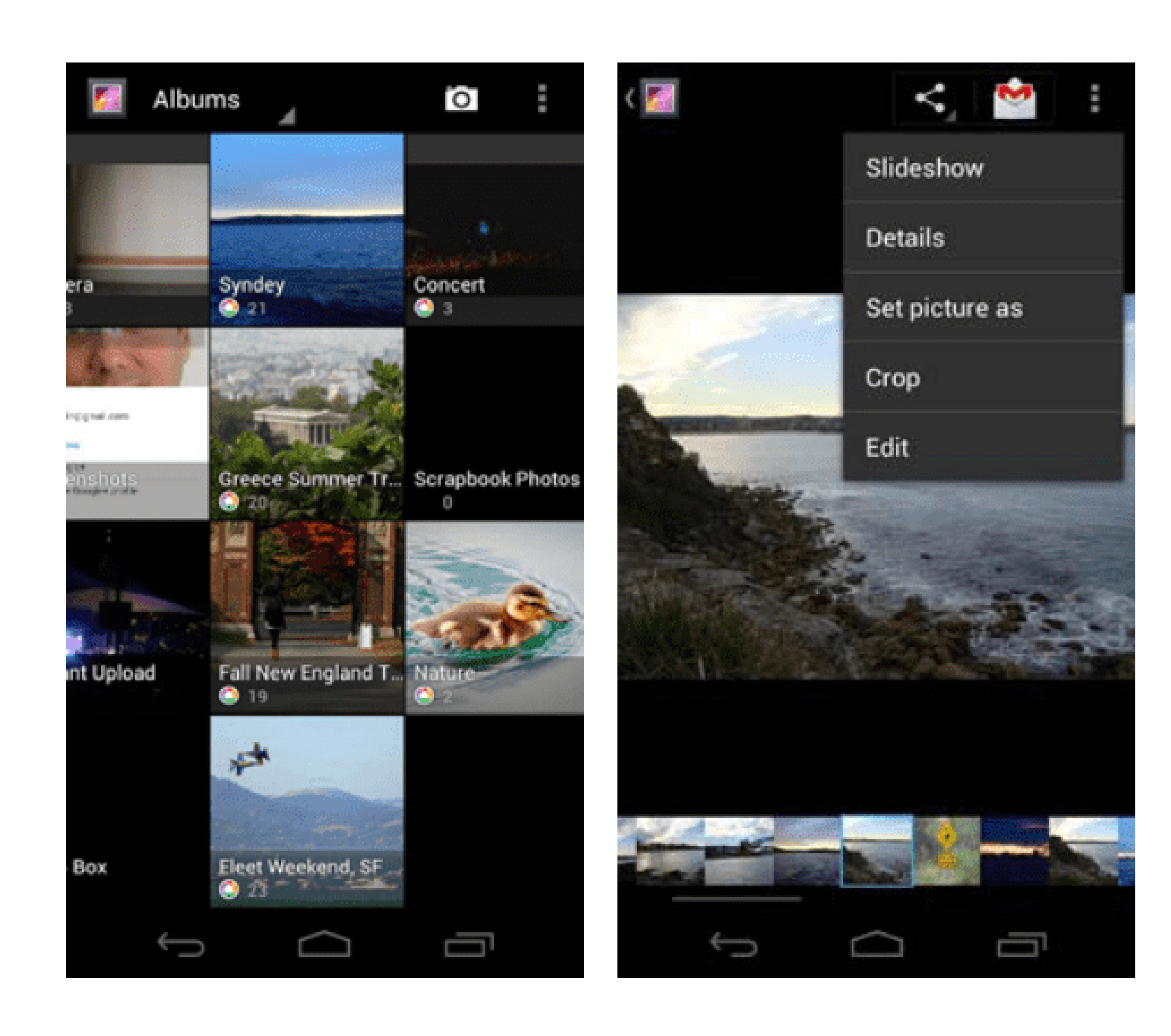
下图分别是两个系统显示当前所打开过的应用的列表。Android 和 iOS 在这个功能上的设计非常不同，也非常有趣。Android 通过 Home 键的长按可呼出应用列表（如左下图），列表中显示的有应用名和应用隐藏到后台时的界面缩略图。顺便提一下，长按 Home 键来切换应用是 Android 测试中很重要的测试。Android 中止处于进程中的应用

的操作更有趣，只需将要中止的应用缩略图右滑即可。iOS 则是通过双击 Home 键来呼出应用列表（一般而言 iOS 的应用列表非常长，只要是用户打开过的应用都会在其中列出，列表中显示应用的名称和图标），中止这些应用的操作与删除操作一致——长按应用，点击应用左上角的停止按钮即可。



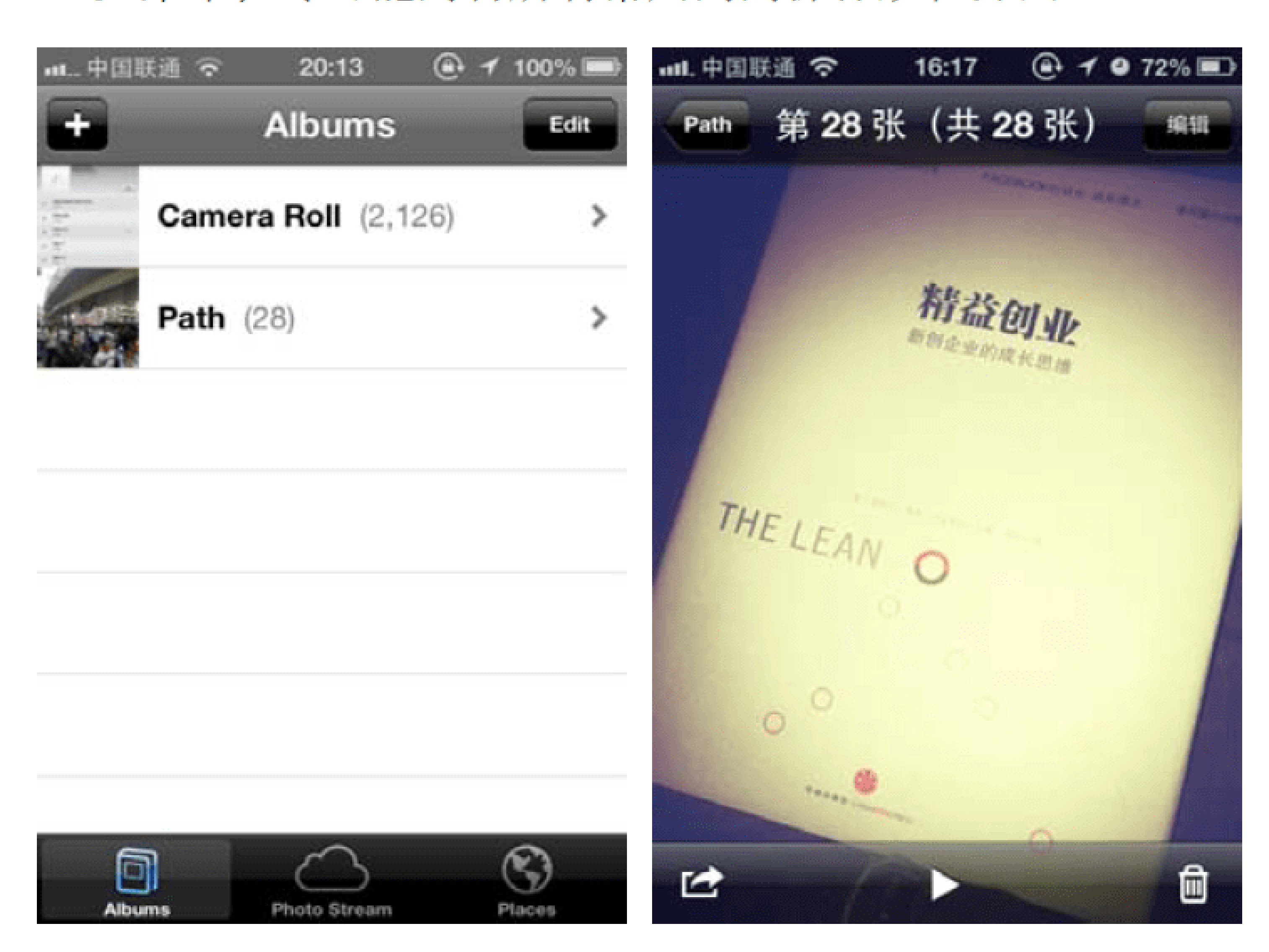
3.2.5 相册

下图分别是 Android 系统预览相册以及选中照片之后的界面。不难看出，Android 贯彻了自己的设计风格，相册的预览是小的缩略图，上方同样显示有扩展功能（如短信编辑界面）的按钮。选中照片之后，返回上一级菜单按钮在屏幕左上角，屏幕右上角的功能也保持了 Android4.0 的设计风格。



我们可以再来看下 iOS 对应的系统预览相册以及选中照片之后的界面。

这两张图分别是 iOS 系统中相册功能和照片选择之后的截图。无论在选择相册还是查看照片界面，iOS 保持右上角的编辑功能以及底部的标签栏的设计风格。从相册这个应用中我们能够看到，iOS 中的设计中并不存在任何下拉式的菜单，同时相对提供的功能也比 Android 更集中，尽可能的将所有常用的的操作放在界面上。



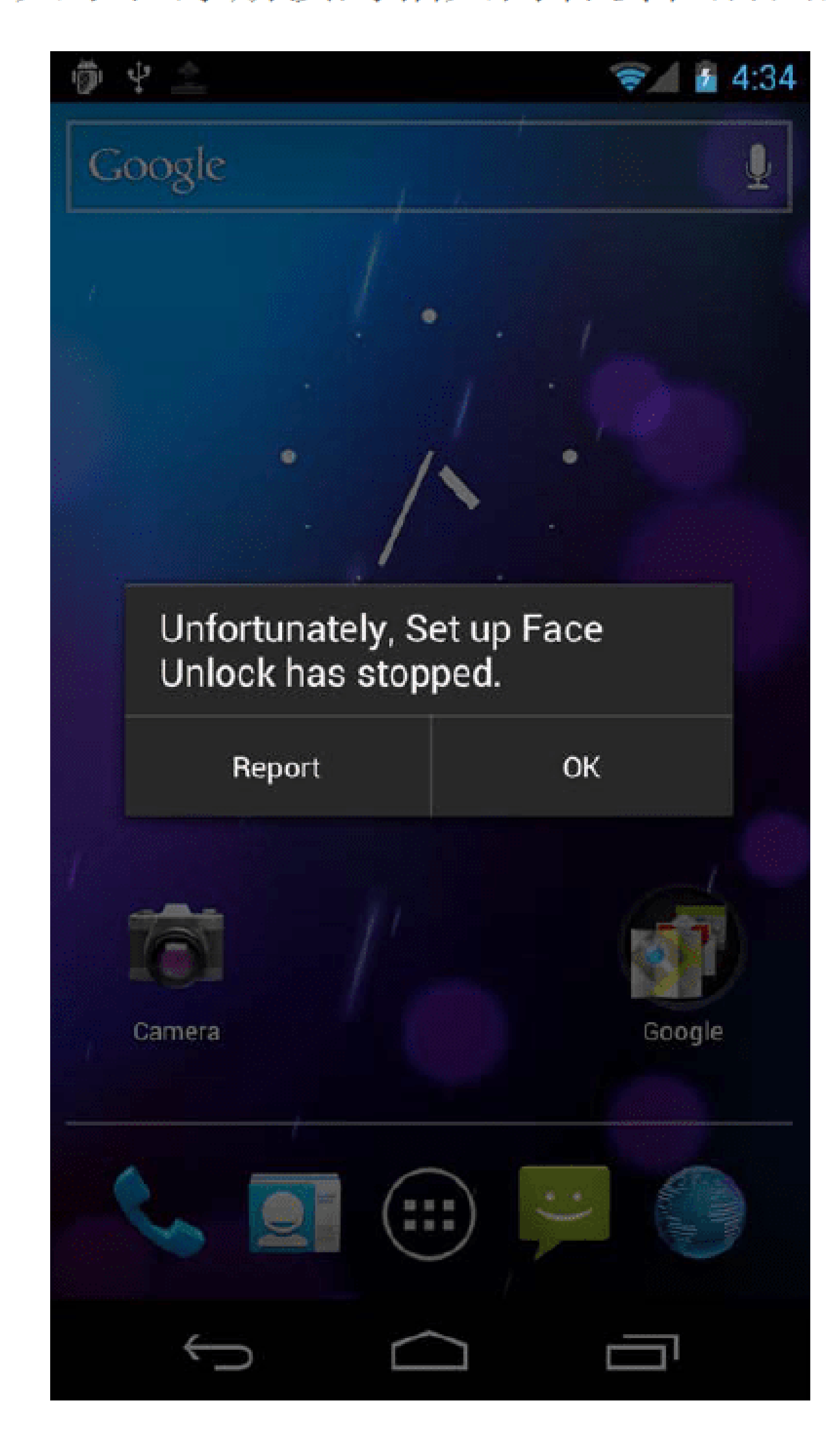
3.2.6 其他

以上通过罗列几个系统应用的界面让大家感受这两个系统在设计风格上的差异，相信大家多少对设计风格应该找到了点感觉了。更多的信息可以由以下网址获得：

Android：http://developer.android.com/design/index.html

iOS：https://developer.apple.com/library/ios/#documentation/UserExperience/Conceptual/MobileHIG/Introduction/Introduction.html

相信做过 Android 系统测试的朋友肯定看到过下图弹出信息。这是在 Android 系统中应用程序崩溃（Crash）的时候弹出的程序中止提示框。在对程序崩溃上的反应，iOS 的表现也与 Android 截然不同。iOS 的应用一旦崩溃之后，会直接退出并返回桌面，不会出现任何的提示信息。关于程序崩溃的相关内容会在后面几章陆续讲解。



Android 和 iOS 的设计区别不仅体现在界面（UI）的设计上，更多的是体现在两家公司的理念以及硬件上。Android 从开始就提倡开源，经过几年之后，现在各种二次开发的 Android 系统层出不穷，比如 CM、小米、锤子等。这些系统的出现对于开发和测试人员来讲根本就是一场噩梦，他们需要保证自己的产品在各种 Android 系统中都能正常运行。iOS 则恰恰相反，不肯开源。在硬件方面，两个系统除了 Home 键、Power 键、音量控制按钮一样之外，Android 智能机在发展过程中还出现过滚轮、方向键、菜单（Menu）键、返回（Back）键，甚至还有全键盘的硬键盘。最新的 Android 智能机依然保存了菜单键和返回键，只不过是从按键进步到了触摸式按钮，这些变化对测试工程师来讲大大增加了测试难度，也让用户的交互操作变得更加难以捉摸。

3.3　“愚笨”的用户——用户引导

假设你很有想法，聚集了一群志同道合之人将你的理念变成了一个产品，然后信心满满地开始新产品的推广。但是往往事实和理想相差甚远，慢慢发现使用产品的人数很少，并且还在不断地流失。在这个过程中，很多员工会骂用户怎么那么笨，明明有那么好的功能，那么多靓点，为什么就是不用呢？事实上用户大部分是非互联网从业人员，需要学会从他们的视角去看待自己的产品。因此，如何更好地引导用户使用并留住更多的用户就显得非常重要。当然，这也是移动互联网测试工程师们需要测试的重点之一。

引导通常分成两种——第一次打开产品的引导和产品功能引导（显性和隐性）。现在的移动应用都会做一套自己的注册系统，让用户注册该产品，从而更长久地留住用户以及获取更多的用户信息。当用户在什么都不了解的情况下第一次打开一个应用，它给用户的第一印象很重要。

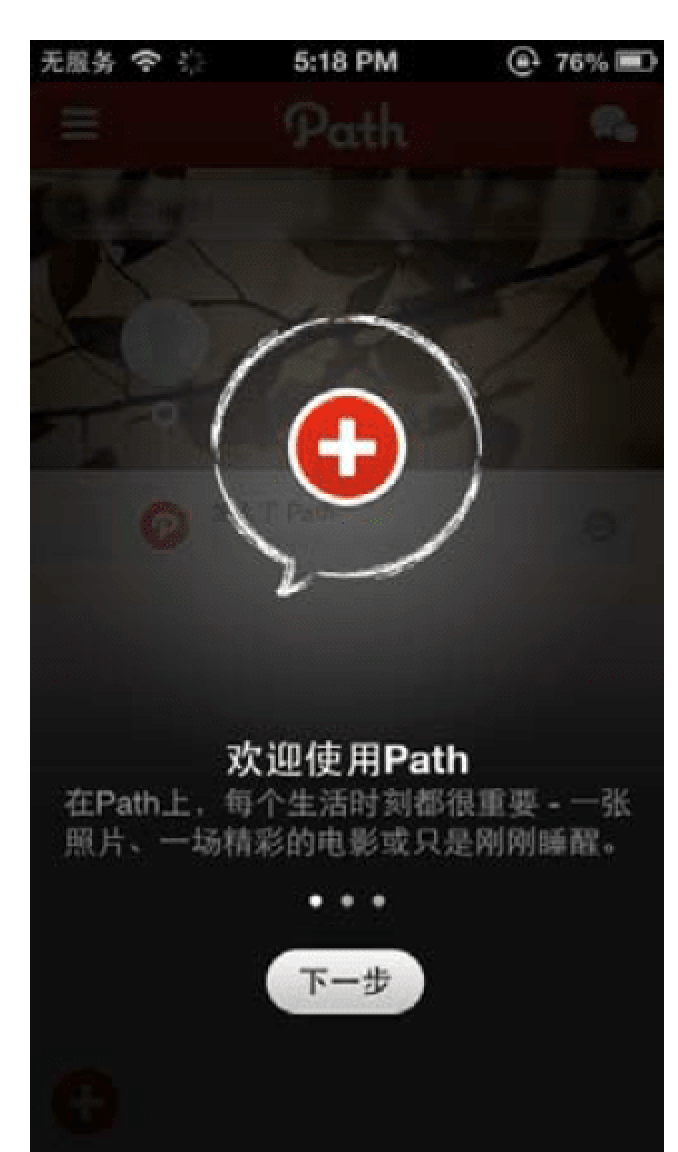
我喜欢 Path，因为它是近几年难得让我欣赏和值得去分析的好应用。来看一下 Path 第一次打开时的界面。

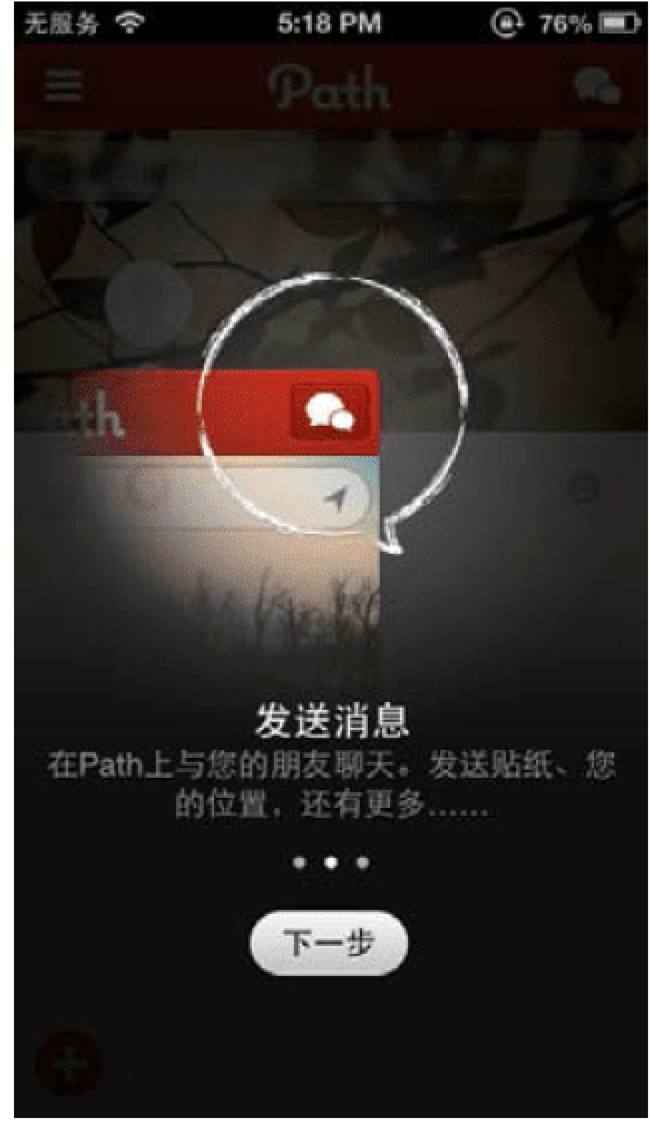


个人认为有以下几个优点：

- 背景优美。5 个滚动界面背景被精心设计过，让人感觉赏心悦目。
- 简洁。整个界面只有两个按钮：“注册”和“登录”按钮，并且都安排在界面的了最底部，从而不破坏应用整体给用户的第一印象。
- 不阻碍式宣传。Path 的受众定位是 150 个人的私密朋友圈，它在 5 个滚动界面中分别简洁地阐述自己的定位，但显示这些信息没有防碍用户进一步的操作（比如注册或登录）。

第一次使用 Path 的用户通常会点击注册，注册完毕后会出现第一次用户使用引导的界面。如下图：



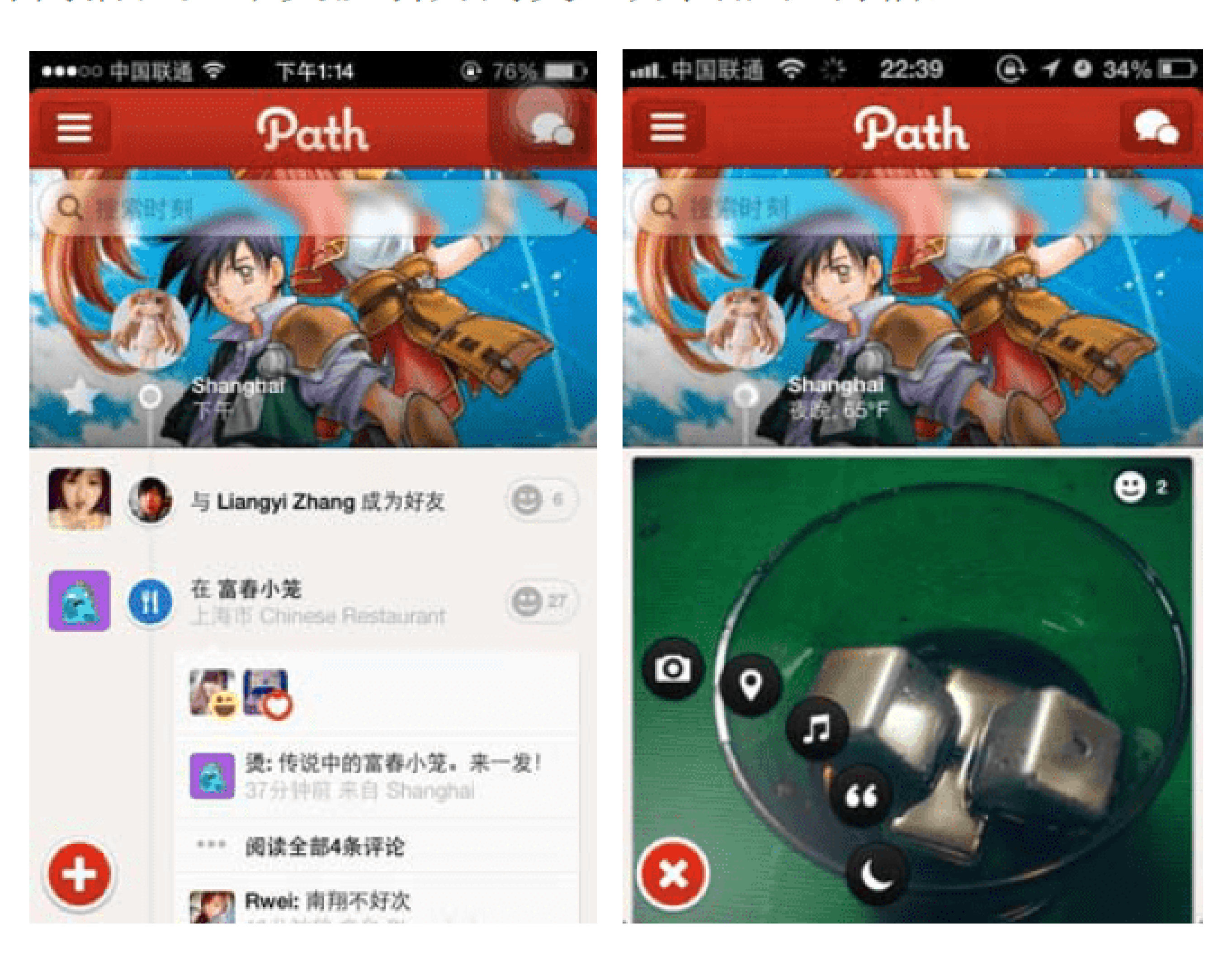




这种用户第一次登录应用出现的引导界面现在越来越多地被使用出现。但是需要注意以下几点：

- 引导界面最多不要超过 4 页。一般而言，超过 4 页的引导对于用户是没有吸引的，大部分用户没有耐心看完这些精心设计的引导页面，他们更想新体验新事物。Path 的引导页面设计了 3 页，个人觉得恰到好处。
- 突出重点。目前 Path 最新的版本是 3.0，其主推的功能是朋友之间的消息聊天。仅用几个引导页突出产品更新的核心功能并不容易，因为如果需要突出的功能点太多，那么这款应用根本就不是一款好应用。
- 零学习成本。引导页要足够简单，让任何领域、年龄段的用户一看就明白，同时也要做到，用户哪怕快速地跳过了引导界面，也能够在使用过程中很快学会使用产品的核心功能。Path 很好地做到了这一点。我们可以从 Path 的主交互界面看到它是怎样简单明了地交互的：明显的时间轴、左右上角的两个功能按钮以及左下角的“＋”，如上图所示。顺便提一下，这个“＋”的设计也是 Path 的创新点之一。

如下图显示的，用户可能没有仔细看过引导界面的文字和图片，但是当他尝试性的去点击“＋”按钮之后，弹出的那些按钮可令其功能一目了然，如下图所示，完全没有学习成本。这种简洁但又不失吸引力的交互方法很值得借鉴。



曾经对移动互联网应用的引导部分的效果做过调查，大部分的用户会快速操作跳过引导部分。即便如此，引导界面还是对用户有一定影响的，比如当看到下面这个引导界面时，相信有人会头皮发麻。





其实以上这类应用还算好的，随着企业想抓住用户，让用户注册自己产品的野心越来越大，很多产品在注册引导方面根本就没花心思去设计。很多应用一打开就跳出一个注册界面，上面密密麻麻的都是需要用户输入的信息。付出和索取是相互的，一个产品还未引起用户的兴趣，未让用户去试用，就想索取用户的各种信息，怎么会被接受呢？

随着移动互联网应用的发展，更多的应用已允许用户使用常用的社交社的帐号进行登录（比如新浪、腾讯微博和 Twitter 等）。虽然这样做可以免去用户重新注册一个新帐号的麻烦，但用户还是免不了要登录一次，对用户来讲是有成本的。所以现在通用的做法是，在用户登录之前开放产品的部分功能，让用户可直接使用。这样，无论登录还是注册都不会成为用户试用产品前必须的步骤。

多数应用的启动都需要一定的等待时间，《互联网时代》曾经对用户做过调查，互联网用户等待网页打开的最长时间是 5 秒。移动而互联网时代客户用户群等待的最长时间已经缩短到 3 秒。尽管如此，在短短的几秒钟内也能大做文章。

一般应用会在用户等待开启期间显示一个包含自己产品 Logo 的清爽界面。如下图所示。







新浪微博曾在开启等待界面增加了一个“东风标致”的画面，这不仅使得用户的等待时间延长，还使得很多使用新浪客户端的用户转而投向了 Weico、ZAKER 等应用。虽然我不清楚新浪和东风标致是否有过合作，但可以确定的是，在新版的客户端中已经看不到东风标致的广告了。相比单一显示产品 Logo 的画面，我更喜欢 ZAKER 和 Linkedin 的启动画面，它们不但突出了自己产品的 Logo，其背景画面也很好地阐述了各自产品的理念。

ZAKER 每次启动会有不同的等待画面，让用户感觉新鲜的同时，也体现了 ZAKER 清新简洁的风格。Linkedin 的背景则是以突出职场的忙碌以及快节奏的社会为特色。

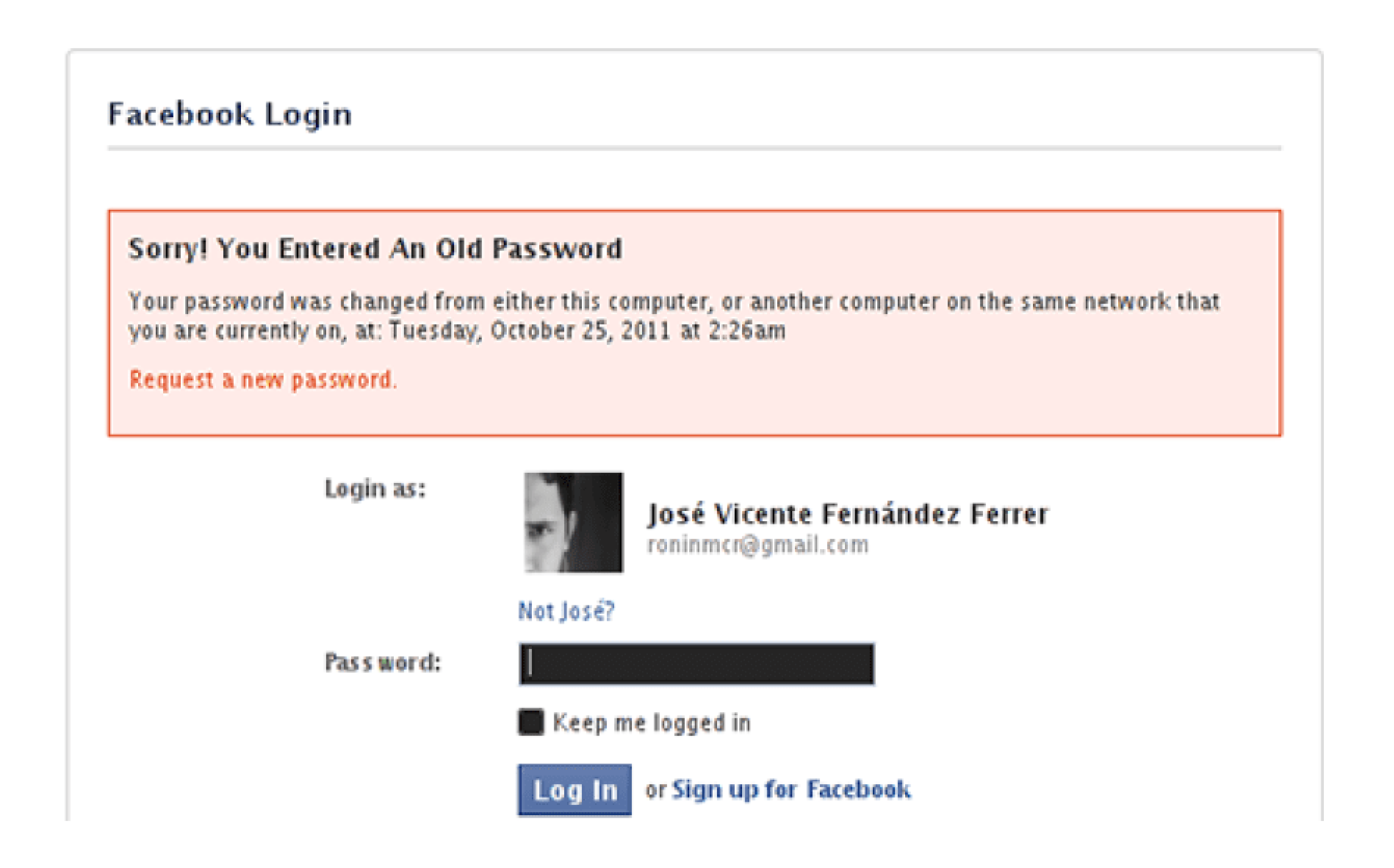
3.4 “捣乱”的用户——应用容错

要想让自己的产品留得住用户，让用户玩得更长久，那么应用就得承受得住广大用户的“故意”捣乱 。用户是一个很敏感的群体，他们一旦发现所用的应用出现崩溃或者自己无法解释的错误，大部分人会选择重新启动。如果还不能解决问题，那么他们就要和这款应用说再见了。只有极少数用户会对问题进行反馈，但他们不关心问题出现的原因和解决方法，如果企业无法解决他们反馈的问题，那么最终这些用户还是会选择永久地删除这款应用。所以，对应用容错的用户体验测试显得尤其重要。

3.4.1 注册与登录

相当一部分应用从注册和登录界面即让用户失去兴趣。我们开发游戏应用的时候做过统计，复杂的密码策略可以导致 50%的用户直接放弃注册，特别是对本来就不出色的应用更是如此。当用户在注册一个新帐号的时候，往往担心用户名是否与其他人冲突或密码的输入是否符合规范。这是因为在智能手机或者平板电脑上精确输入的成本很大，很容易由于出现错误导致前功尽弃。用户当然不愿意看到自己辛辛苦苦填写完所有需要的信息之后，软件提示输入错误，要求用户重新输入（有些应用甚至会清空已输入的字符）。所以，应用需要在错误发生的时候保留用户已经输入的正确数据，同时给出

确切的错误提示信息。例如 FaceBook 的提示：



在注册的过程中还有一个要素是用户关心密码的安全级别。在传统的网页上，系统能够很好的告知用户。

Password　6 characters or more (be tricky!)
Password　Too short
Password　Too obvious
Password　Weak
Password　Good
Password　Strong
Password　Very Strong

在移动客户端应用中如果也采用同样的设计，就会显得与整体不协调，有一种很突兀的感觉。在这方面，tumblr.的设计很人性化，当用户输入不同位数，由不同字符组成的密码的时候，输入框的右方会有类似于“能量条”的指示栏出现，这样的设计通俗易懂，大大降低了用户的学习成本。



3.4.2 断网引发的问题

在中国，移动互联网用户还有一个经常遇到的老问题——网络问题。网络问题直接影响了一个应用的体验好坏，目前，各个应用基本上都需要连接到网络，大到网络游戏，小到在线升级等等。这样的变化是必然的，应用网络化可以大大减小应用程序本身的容量，同时也能在不更新客户端的情况下更及时更新网络数据。但是测试人员在测试应用的时候往往会遗漏一些与此相关的测试切入点或用户体验问题。

比如，一些应用在无网络的情况下启动会提示没有网络，但是界面上看到的数据还是很完整的。



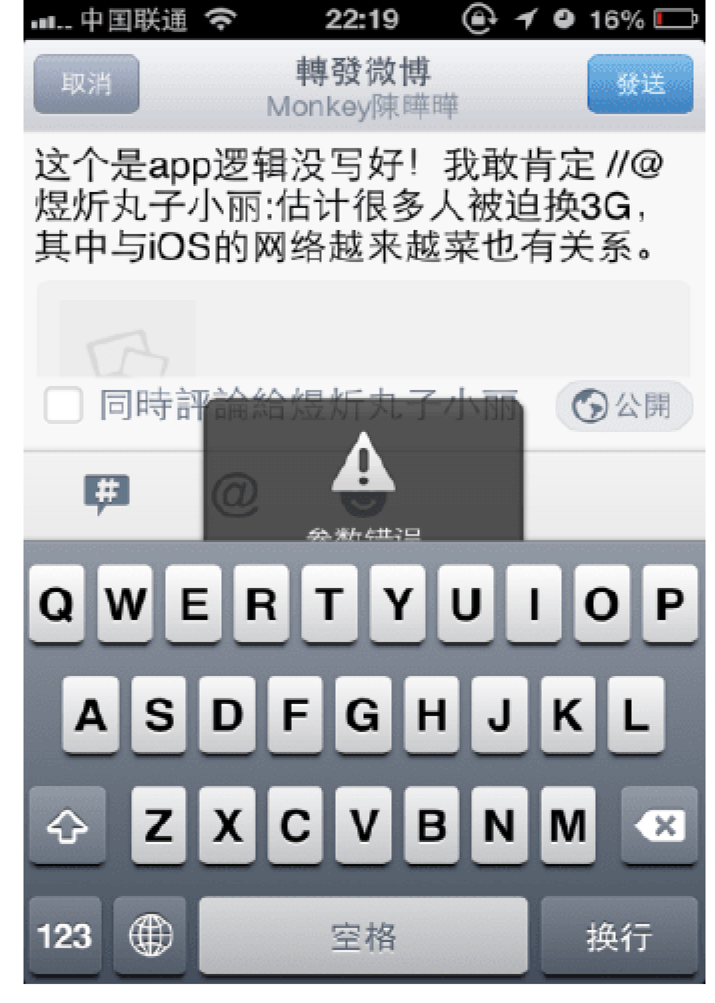
这并不表示这款应用在无网络连接的情况下也能正常运行。之所以能够看到数据是因为这款应用曾经被启动过，显示的这些数据只存在于缓存中而已。那么问题就来了，这些应用刚下载完毕，第一次在无网络情况下打开会不会正常运行呢？我自己随机测试了一些应用，事实证明，在同样的环境下（iphone4，iOS6，无网络），包括联通营业厅在内的多个应用一启动就崩溃，根本无法使用。这种体验能让不用户抓狂吗？

在没有网络的情况下，大部分应用都能够提示用户没有网络连接或类似的信息。



但在网络较差的情况下，很多应用给出的提示就开始“各显神通”了，从专业角度来讲，这会让用户体验变得非常糟糕。我们不妨来看一下几个应用的提示。







第一幅图是新浪微博最新的iOS客户端。以前我个人还是很喜欢新浪微博客户端的，自从更新之后我发现吐槽这个客户端的微博不在少数。不过值得称赞的是，新浪草稿箱对未发出的消息做了错误消息的归类，这样用户就可以一目了然，知道发送不出去的原因是什么，可惜对发送错误原因的判断很容易出现误差。比如第一幅图显示的，由于网络错误导致发布失败的微博都显示了“20101 微博不存在哦！”字样，对用户来讲微博不存在是一件相对严重的问题，更何况“20101”又是什么呢？相信大部分用户是无法理解错误信息代码的。

第二幅图就是点击第一幅图的第 1 条信息之后出现的发布微博画面，可以看到点击进入之后直接提示“参数错误”，对用户说“参数错误”无异于说火星语，用户不懂什么是参数，也不懂为什么错误，他们只知道自己的微博发不出去了。如果长此以往，他们会放弃新浪客户端甚至新浪微博。

第三幅图是新浪微博另外一个客户端 Weico。就如之前说的，在一个应用（新浪客户端）无法正常使用（发布微博）的情况下，用户会转战到另一个应用（Weico 客户端）。让我惊讶的是，在网络不稳定的情况下居然提示“微博 API 连接超时”。如果我是一个用户（实际上我就是一个用户），可能还能理解“超时”，但是请问“API”是什么意思？

以上三张图的提示否非常的“奇葩”？事实上，应用的发布版（Release）和调试版（Debug）毕竟还是要区分开来，如果容错提示语言是用户无法理解的，那还谈什么用户体验呢？

3.5　专业精神——风格一致性

当我们要着手去做一个应用的时候，在让用户看到一个我们要表达的理念之前，先得让用户感受到我们的专业性。这通常表现在两点：

- 应用的设计风格要和所在系统设计风格保持一致。
- 应用本身的设计风格需要保持一致。

3.5.1 应用与系统风格一致

首先说第 1 点，保持应用与系统风格一致，这是什么意思呢？比如说一家企业在 Android 或者 iOS 上做了一个应用，这个应用所有的交互方式是早期诺基亚时代的交互方式，而且画面都是粗像素颗粒化的，这样的应用能算一个好的应用吗（当然，市场上具有一些怀旧风格的应用或者游戏，但是其目的就是怀旧，不是主流）？

我再来说个正常点的例子，本章第 3.2 节将 Android 和 iOS 两个系统的风格做了简单的对比介绍，试想，一个基于 Android 系统的应用如果全部采用 iOS 的风格设计，大家肯定觉得别扭。从现在的应用市场来看，也没有哪个应用是这样做的，也不会这样做。这就是说，设计的应用需要匹配所在系统的设计风格。Evernote 在这点上做得特别出色，我们来看看它在 Android 上的界面设计和交互。







Evernote 在基于 Android 系统上的设计很“Android”。我们能够看到它有暗色系的设计、熟悉的搜索按钮和下拉菜单设计等，都中规中矩地参照了 Android 系统的设计风格。同时 Evernote 引入了现在很流行的一种“侧边栏”设计，用户能够在不切换界面的情况下看到更多的功能或选择项。该设计因 Path 的发布而大面积的风靡。

Evernote 在 iOS 系统上的设计与 Android 系统上的大相逕庭。在 iOS 环境下，Evernote 采用了文件夹式的设计，主色调为鲜艳的绿色并去除了“侧边栏”，保持自己的风格。



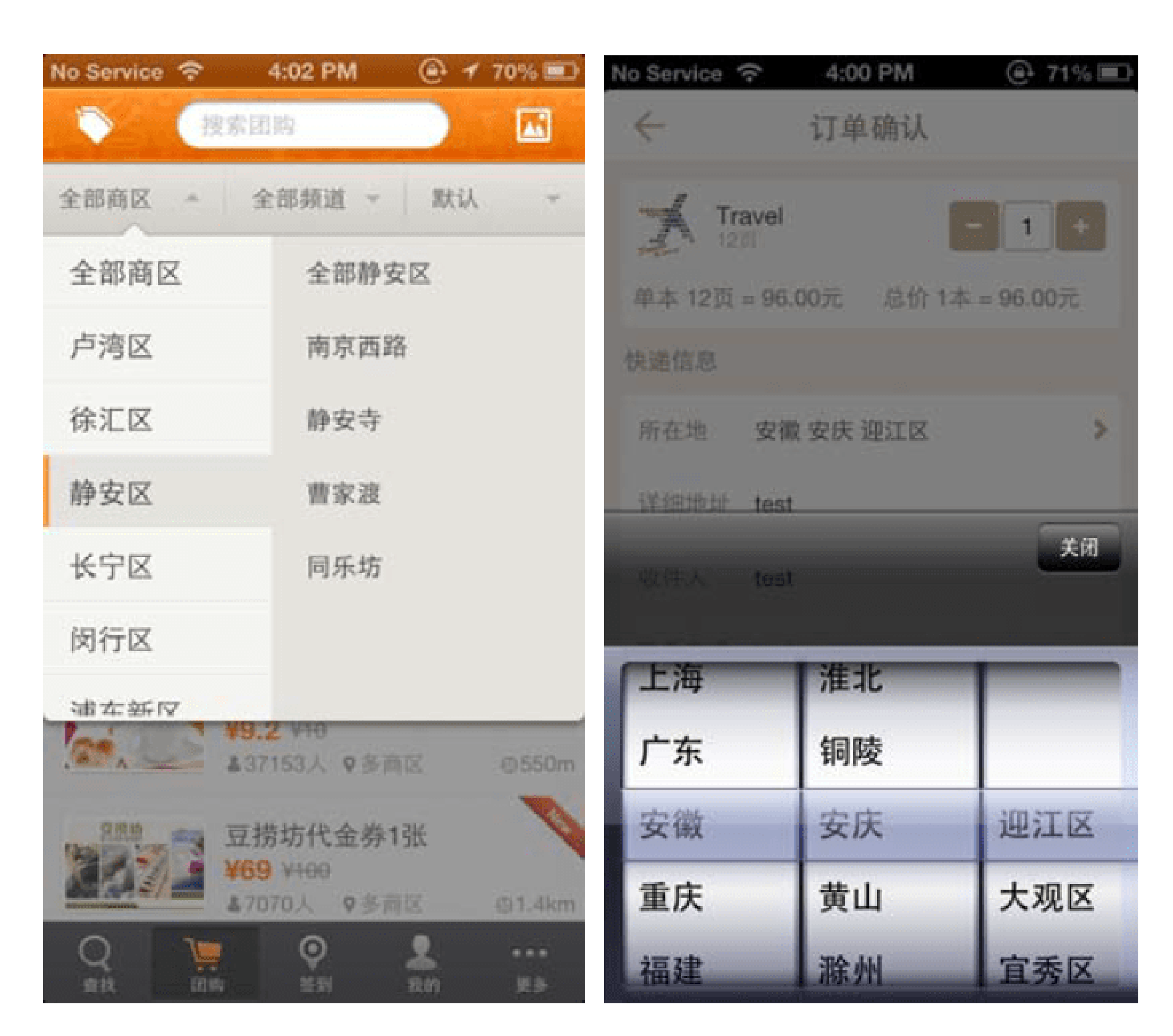
3.5.2　应用本身风格一致

其次我们来说一下应用本身的风格需要保持一致。一般来说，这类风格的不一致出现在不同的界面以及细节处，比如：

- 一些返回按钮的设计。是否会出现有的界面是按钮设计，有的界面是图案设计。
- 弹出提示框的设计。是否会在同一个应用中出现两种不同风格的提示框。
- 下拉菜单设计。是否会在同一个应用中出现两种不同风格的下拉菜单。
- 颜色。是否会在整体暗色调的应用中突然出现一些很鲜艳的亮色系的颜色。

- ……

在一个应用中，如果出现不同的设计风格，从功能上来说不会造成任何影响，但是从用户体验上就会让人感觉非常不专业，是一个“山寨”应用。这里给一个下拉菜单风格的例子。



这两个应用界面中。都实现了不同风格的列表功能。试想一下，如果一个应用中同时出现两种以上不同的列表样式和风格，你会有什么感觉呢？

3.6 “我”即最终用户：过程体验测试

作为一个测试人员，应该像真正的用户一样去使用应用。假设用户已经跳过了复杂的引导界面并且也不关心设计风格，那么他们一定得学会在使用的过程中体验这款应用。

每款应用都会有自己的核心功能。比如微博是一个社交平台，发布微博就是它最核心的功能之一。假设我们使用新浪微博客户端发送一条微博的流程是这样的：

（1）点击编辑按钮；

（2）编辑文字；

（3）上传图片；

（4）选择所在地点；

（5）选择发送微博所针对的分组；

（6）其他操作。

那么新浪微博这个产品估计已经不存在了。很多企业做移动互联网的产品时都喜欢向微软学习，什么功能都想往产品里加，什么功能都想做好，最后这款产品就会变成四不像，没有自己的特点。移动应用的核心功能应该越简单越好。

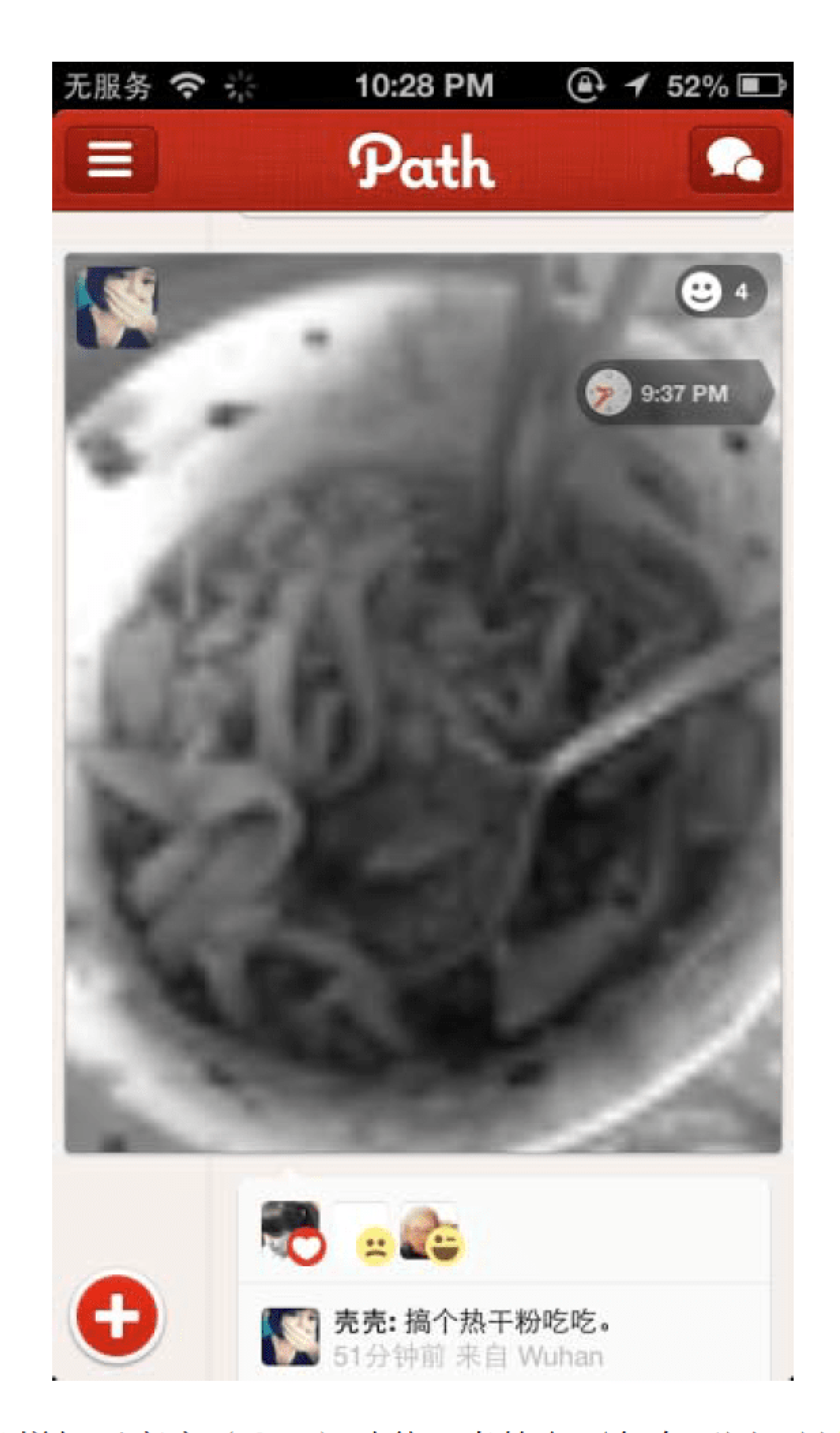
我们拿社交应用来举例吧。用户在使用中有两个重要的过程——发送消息和接收消息。发送消息的过程一般都需要填写相关信息，可能是你想让别人知道的信息，也可能是你不想透露的信息。对于接收信息也是一样的。但是由于网络因素，有时发送和接收就变得不那么容易。我们在非 Wifi 的情况下，不得不面临等待、稍等再发、网络错误，甚至应用崩溃等各种突发情况。这时，如何让用户在无聊的等待之后还能继续使用这款应用就成了很多设计师设计时要考虑的重点。

很多社交应用离不开图片，而等待大量图片载入就成了每个用户的必修课程。



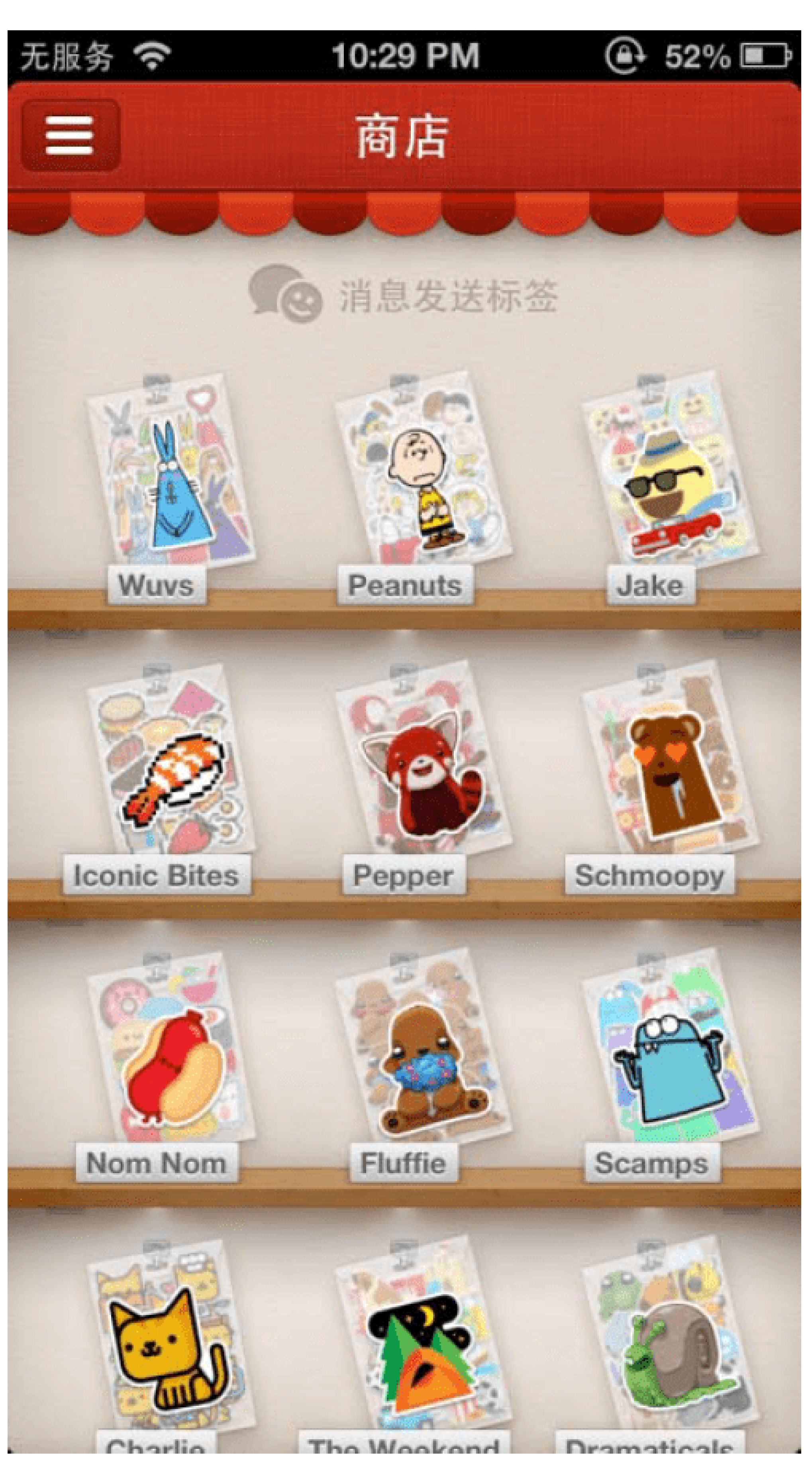
有的应用会像左上图那样的设计，在网络差或未加载完图片前给用户显示一个 ImageView（Android 默认的一个图形控件），也就是一个灰色底的框框。还有的应用采用右上图那样的设计，在图片加载完毕之前给用户显示一个预设的背景图片，以缓解用户看到灰色底框时的莫名心情，通常背景图片是一个设计过的图片或是 Logo，也可能是“等待中”字样。其中有一款应用在这一点上做得让我赏心悦目，估计已经有朋友猜出是哪款应用了——正是我钟爱的 Path。

Path 在图片完全加载出来之前，先加载部分数据，以便让用户有一张模糊的图可看。这样，在图片完全载入之前，能够让用户看到图片的大概内容，使得用户的等待不会太无聊，还增加了猜测图片的乐趣，至少我自己是这样的。



现在很多应用都增加了商店（Shop）功能，卖的东西各有不同，目的只有一个——赚钱。用户买东西也是一项很重要的活动，淘宝在这方面就做得很好。在移动互联网应用中，买的东西可能是一些软件或者游戏插件，它们不需要用户填写复杂的地址和邮编，相比电脑上购买实体物件已经非常简便。在此背景下，各个“商店”都需要一些创新点来吸引顾客。不得不说，现在的社会真是一个卖萌的社会，各种软件都卖一手好萌，但在细节上，Path 又领先于其他应用一步。





进入 Path 的线上商店，任谁都会被可爱的卡通设计所吸引。商店主要卖私密好友间聊天用的一些可爱的表情，或者一些聊天背景风格。这些商品的标签就如现实生活中一样被粘在展架上。当用户在预览商品时倾斜智能机的话，惊人的一幕出现了——商标如同感受到倾斜般也跟着一起晃动。虽然应用借助智能机的陀螺仪要做到这点并非难事，但对于用户来讲绝对是一个创新点、一个吸引点。这个设计之后也出现在其他应用中，比如随享新版的登陆界面，其绿色草地背景可以根据手机的倾斜度看到周围扩展的景色。



总结一下，在移动互联网应用开发过程中，测试工程师们要比项目中的任何一个人都要多地使用应用，体验应用。可能在测试过程中已经用得厌烦了、觉得枯燥了，但依然需要从不同的角度去体验应用的使用过程，找出可能会让用户感到困惑、厌烦的地方，这也正是我们测试工程师的职责。

3.7　使用更多的应用：对比体验测试

测试工程师们往往长期从事固定的业务和应用，一段时间之后，大多数人都会出现固化的思维以及厌卷的心理。此时很多测试工程师会说：

“我发现我很难再找到缺陷了”，“这个应用我已经非常熟悉了”。

我们每个人一生中，为企业付出劳动开发的应用可能是有限的，但是能使用的应用是无限的。在我们测试一款应用的时候，最好的一种测试方法就是，在同样的测试环境下,去使用同类型的其他应用,这样很容易就能看出自己测试的这款应用设计上的缺陷。使用更多的应用能够很好地帮助我们提升用户体验。

例如我自己是做拍照、图片社区的应用测试的，长期以来我会养成以下习惯:

- 第一时间使用同类应用的最新版本。比如啪啪、图钉等。
- 定期使用 Google Play 和 App Store 上推荐的和最新的拍照及社交应用。
- 定期观察用户对于自己和同类应用的反馈。



提示：在这里还有一点不得不提，我们大部分测试工程师都在国内工作生活，所看所用的应用也都是国内的。但是，很多国外的软件都有着很强的前瞻性和很好的创意，需要定期地多体验体验国外的应用，学习它们的一些内涵和创意。正所谓“知己知彼，百战百胜”。

3.8 模拟场景体验测试

软件测试中除了功能测试、压力测试、性能测试等大家耳熟能详的测试类型外，还有在移动互联网应用测试中尤为重要的一个场景测试。顾名思义，场景测试就是测试工程师需要模拟用户使用应用时候会遇见的真实场景，尽可能发现应用中的问题，从而保证应用的可用性。

在我们进行场景模拟测试之前，首先需要弄清楚应用的受众群体处于哪个年龄段，什么性别，从事何种工作等信息。

虽然很多管理者都知道应该做减法，但有很多企业做应用还是采用“微软”风格——不停的增加功能。很多应用天生定位就是模糊的，导致项目后续困难重重，员工连连叫苦，老板抱怨员工效率差，然后、然后就恶性循环了。

测试人员要模拟最终用户去使用应用，前提就是要知道最终用户是谁。现在如果移动互联网应用的定位不准确，那么几乎是要被埋没的。放眼望去，做得好的应用都是有明确的用户群和市场定位的。例如：

- Path——私密的朋友圈，限制好友 150 个。分享状态、图片，聊天等。
- Vida——主要针对年轻人，专业做照片分享的社区。
- 陌陌——针对年轻人群，交友。
- Linkedin——主要针对商务白领和猎头，寻找人才或工作。

知道了主要的功能以及针对的人群，才能够准确地了解这些人的心理、生活习惯、工作场景等等，才能够更好地进行模拟测试，否则一切测试活动都是在扯淡。

场景测试就是一种模拟用户实际会使用到，会遇到的场合进行测试，而非自由测试或者根据测试用例进行测试。我们来看两个真实的应用进行场景测试的例子吧。

3.8.1　应用一：智能手机输入法

针对机型：Android 和 iOS 系统的智能机

针对用户群：所有使用智能机的用户

模拟场景测试举例：

- 用户正在进行整句的拼写输入，但是发现中间有若干错误，输入法需要有能够让用户以简单的方法修正错误又不影响本次整句输入的其他内容的功能。
- 用户需要快速地、正确地进行日常整句的输入（包括中英文混合的情况），比如“我正在和我同学一起在 11 上打 Dota”。

- 用户平时会在各个不同的应用中输入不同的信息，包括正常输入、密码输入、网址输入等等。
- 用户需要能够在日常使用的软件中正常输入，而不会有任何不兼容的问题。
- 用户进行输入的时候允许出现弹出系统提示框或者手机来电、待机等各种并发情况。
- 输入法需要根据所安装的系统语言进行默认启动语言的自动切换。
- 输入法键盘会挡住智能机屏幕上的部分信息，是否提供了可让用户能够方便地隐藏输入法键盘的功能。
- 九宫格、全键盘等需要根据用户使用习惯的不同而调整出词的优先级。
- 如果是老年用户，需要提供大号的键盘和字体；如果是年轻用户，需要词库提供一些热门的电视剧、动画的术语以及一些常用的可爱表情。
- 有的用户使用的智能机带有硬件键盘，那么输入法也需要正确地支持这类硬件键盘。

3.8.2 应用二：智能机顶盒

针对系统：Android/OS X/Linux

针对人群：长期在家观看电视的人

模拟系统升级场景举例：

- 用户使用网络环境各有不同，PPPOE、WIFI、动态分配等。
- 观看电视一般是中老年人占多数，在输入验证码或者密码的时候，需要提供更方便的输入方法。
- 在观看影片的过程中，既使弹出升级提示也能够正常升级。
- 用户可能使用在线升级，也可能使用离线升级。在离线升级的情况下，需要

考虑到用户使用的外接设备的格式，FAT/NTFS 等。

- 升级的过程中出现断网的情况。
- 成功升级过后，用户能够继续之前的观看记录进行观看。
- 用户升级完毕之后可能会使用关闭电源的方法进行重启。

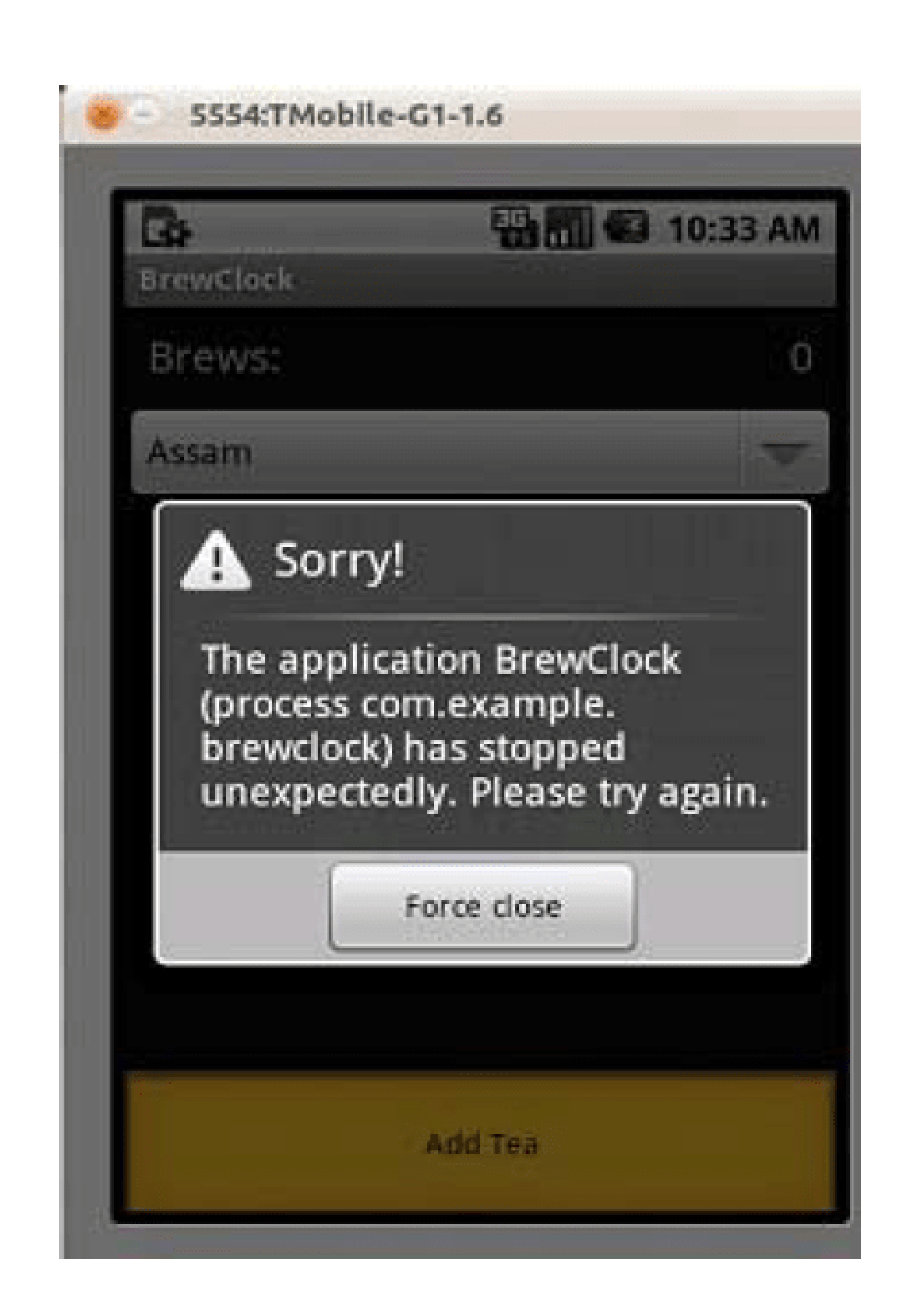
综上所述，我们必须先了解清楚针对的用户群体，包括他们的年龄、生活习惯特征等。然后才能够很好地进行场景用例的模拟设计。从实际项目来看，场景测试用例必不可缺，否则单纯的功能和性能测试就会变得苍白，这也正是移动互联网应用测试与其他产品测试不同的地方。

3.9　用户究竟关心什么？

在移动互联网中有着有各种爱好的用户，从他们使用智能机的系统就能够看得出来，苹果、原生安卓、三星系统、HTC sense、魅族、小米、锤子等等。这些用户在使用智能机的过程中关心的点也会不同。应用需要经过测试的最终目的是让用户用到更好的产品，而用户究竟在想些什么，究竟关心什么则是我们测试工程师需要关心的。

3.9.1　用户只关心应用能在自己手机上正常运行

看上去这像是句废话，不过就这样一句“废话”对目前的移动互联网应用而言都很难被满足。Android 的碎片化和 Apple 的越狱都让“不能运行”现象屡见不鲜。无论进行过怎样的测试，在某些用户的手机上还是会出现崩溃或其他缺陷（见下图）。对这些用户来讲，这个应用非常的垃圾，连自己最基本的使用需求都无法满足。结果可想而知，删除之外，还要补上差评。

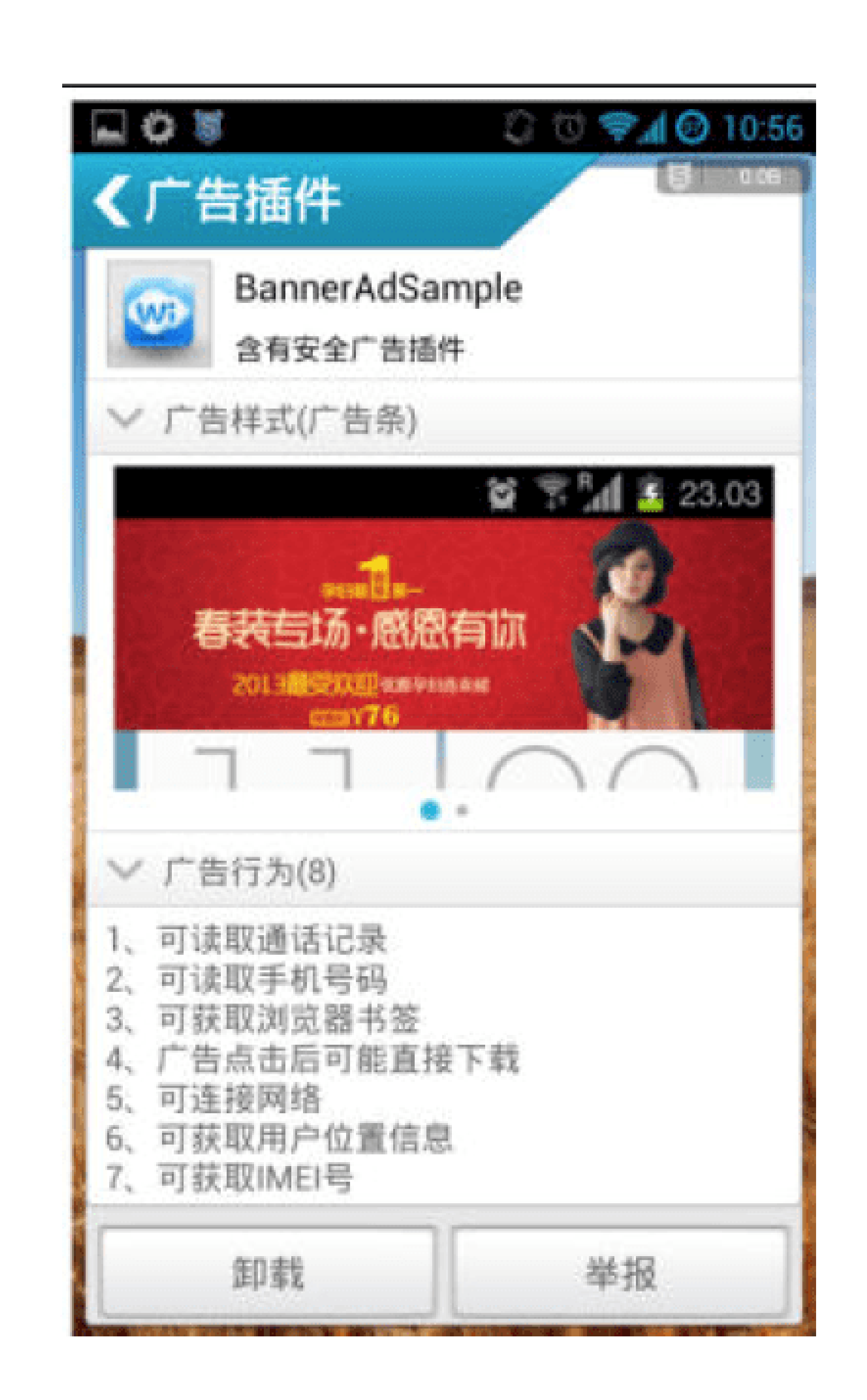


3.9.2 用户隐私权限

现在很多应用都会偷偷地在后台收集用户的数据，这种行为几乎已经变成行业的一个不成文的规定了。搜集数据的目的有的是为了更好地改善自己的应用，也有的是谋取暴利。一些应用的后台老板会说："我们偷偷进行数据收集，用户怎么会知道呢？"

事实上，随着移动互联网应用的快速发展，类似于 360 这样的手机管理软件越来越多。当用户安装这类软件之后，就能够查出自己安装的应用，都使用了哪些权限。当看到安装的应用启用了与自己功能无关的权限的时候，用户肯定会着急甚至愤怒。也许有人会觉得太夸张了。夸张吗？这是关乎隐私的问题，我相信谁都不愿意穿着"皇帝的新衣"在马路上走吧。

下图所示为被查出的窃取信息行为列表。



3.9.3　简洁、方便

我可以很负责任地说，相当一部分用户是小白，部分用户甚至弱智到不能再弱智了，因此，我们的应用就必须简洁和方便。这个问题在前面介绍引导功能中就提到过，无论是引导界面还是核心功能都要让任何背景的用户明白，并且设计要简洁。这样用户才会有利用碎片化的时间去使用应用的欲望。

3.9.4　消耗

说到消耗，可能大家第一个想到的就是应用的收费，除了某些知名游戏以外，目前大部分收费应用的价格还是相对正常的。不过用户金钱上的消耗并非只存在于应用的收费上。在移动互联网发展的初期，大部分的 Android 用户会比较在意一个应用的大小，一个 20MB 甚至更小的应用会直接让用户崩溃掉。而现在用户更多地关心应用本身对

于系统的电量以及网络流量的损耗。这些损耗看似微小，但从长期来讲，对用户是笔不小的支出。

3.9.5 好不好用

对用户来讲，好不好用分为 4 个等级，分别是：

（1）能够正常启动——能用。

（2）能够满足用户的需求，完成特定功能——好用。

（3）能够满足用户的需求，达到了用户的期望值——很好用。

（4）能够满足用户的需求，达到了用户的期望值，还为用户想到了用户自己没有想到的——非常好用。

大家可以将自己测试的应用对号入座，个人觉得大部分的应用都处于 1 与 2 之间，估计能达到 2 的都很少。

其实从用户的角度去看问题的时候，发现用户的要求真的不高。用户从成千万的应用中选择了某企业的应用，安装在自己的智能机上想进行使用，而这个应用却各种复杂，各种崩溃。如果你是用户是不是也无法接受呢？当如此简单的需求都无法满足的时候，用户删除这个应用也无可厚非。

3.10 用户体验的问题是 Bug 吗？

测试工程师小陈：我觉得这个功能的体验不好，参照用户的习惯以及其他应用的做法也许应该那么做。

测试工程师 A：这个是体验问题，不是缺陷（Bug）。

开发工程师：设计就是这样的。

项目经理：体验的问题优先级不高，先放放吧。

??

当测试工程师一旦进行了用户体验相关的测试活动之后，那么上面这段对话几乎是必然会发生的。甚至测试工程师自己也会默默自问：“用户体验的问题到底是不是 Bug 呢？”，我的答案是：“肯定是 Bug”。

移动互联网用户的关注点首当其冲的就是应用体验。一款应用不会因为没有一个 Bug 而闻名于全球，但绝对会因为很好的用户体验而惊艳四座。如果测试工程师通过合理的判断得出在某个功能点上用户体验有问题，那么该问题就必须记录于缺陷管理系统中。不过，用户体验的问题和一般的缺陷的确是需要区分开来的，我们可以通过 Bug 的类型、重要等级、优先级等附加字段来进行区分。任何细小的问题都需要记录在缺陷管理系统中，这是测试工程师的天职。

也许会有很多测试工程师说，自己记录了很多 Bug 和用户体验的问题，但是公司从来都没有空余时间去修复这些问题，感觉提了也是白提。测试不是以某一刻或某项目周期为单位的活动，而更多的是一种长期、迭代的过程。在多个迭代周期后，我们能够更好的分析风险高的功能模块，制定更好的测试策略，有针对性地改进测试流程等等。所以，很多缺陷或与用户体验相关的问题并非一定要在一个迭代内全部解决，但是一定要记录在缺陷管理系统中，否则过了几个周期之后谁也不会记得这个问题了。作为一名合格的测试工程师，我们可以积极地在系统中记录着这些重要但优先级相对不高的问题，主动在报告以及项目中去告知团队的其他成员，存在这些从上个周期遗留下来的问题，并确保最终能够被修复。

3.11　如何提升自身的用户体验经验?

本章说了那么多与用户体验相关的问题，那么到底应该怎么提升自己的用户体验呢？我总结一下主要有以下 4 点。需要注意的是，自己必须实践，必须进行数据搜集，不要自己拍脑袋主观臆断！

- 使用大量的应用。好的或不好的，国内的或国外的，小公司的或大公司的都要使用。在使用的过程中细细体会设计和功能上的优缺点，同时结合自己的应用进行思考。
- 让现实中的用户来使用。找自己的朋友，找非互联网行业的人，甚至可以找不认识的路人甲。记得不要有任何的提示，否则会让他们有先入为主的观念。要真实地收集用户的反馈信息。
- 在应用中添加数据统计的功能。自己的应用每天活跃度是多少，用户经常使用的智能机是哪些机型，用户经常什么时候集中使用应用，应用的哪些功能使用次数最多等等。这类数据都应该统计出来，不仅有助于提升用户体验经验，对定测试策略也有着很好的帮助。
- 适当地阅读一些设计方面的书籍，和交互设计师进行沟通。

用户体验是一个没有定论，无法量化的概念，但却是移动互联网用户最最在乎的点。现在的很多测试工程师往往关注于自动化测试工具、薪资等等，反而不花时间思考自己要学点什么，要积累点什么。所以才会有大量的测试工程师觉得用户体验离测试很远，离自己更远。

3.12 小 结

在移动互联网中，用户体验对应用来讲异常重要，这是有着传统互联网经验的“老鸟”们无法体会的。通读本章，读者多少能对用户体验有一个认识，多少能体会到用户体验的重要性。但是，这些只是冰山一角，还有更多的内容需要经过漫长的学习以及经验的积累，才能向移动互联网资深测试工程师迈进一大步。

第 4 章　功能测试要点

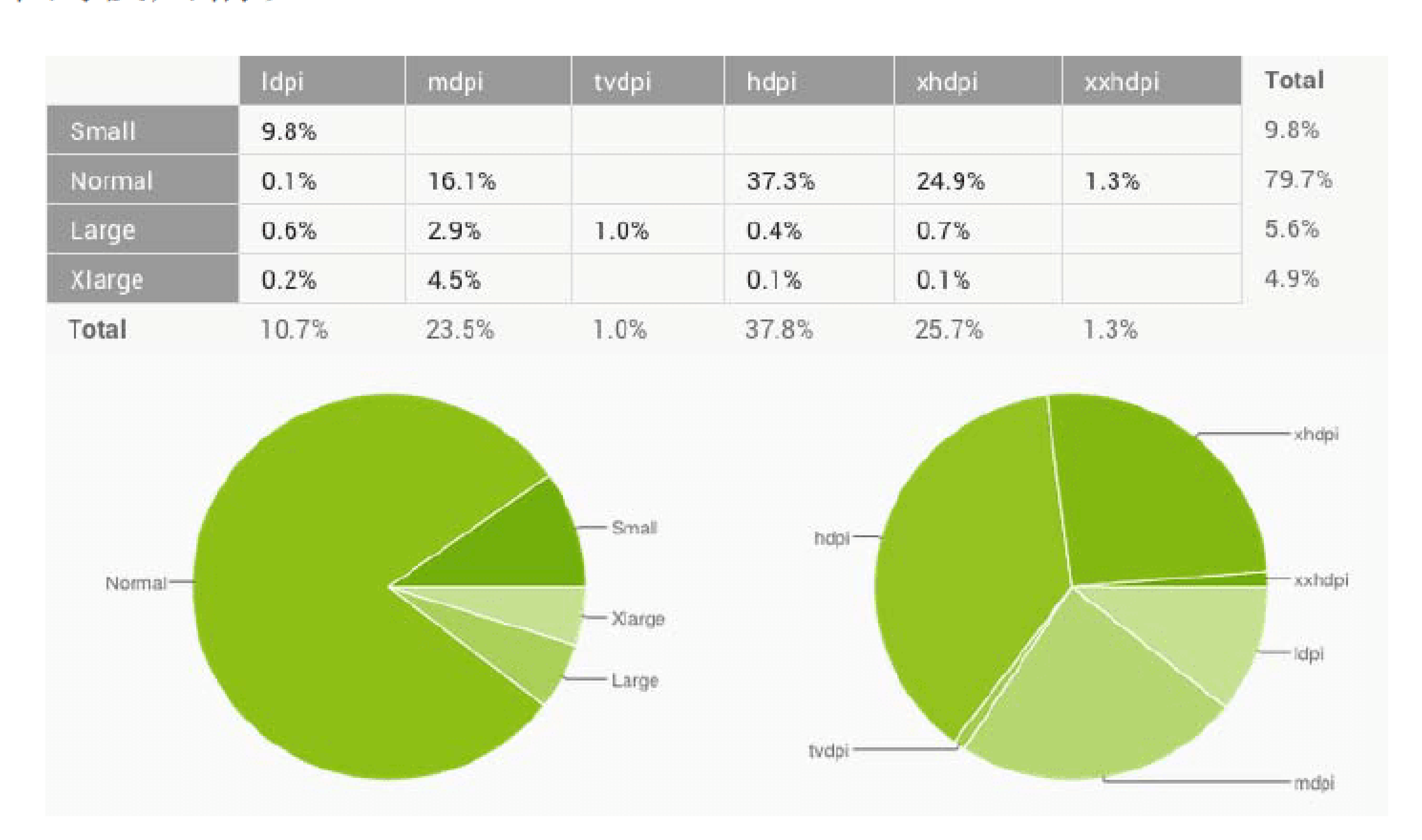
测试工程师经常被问到的一个问题是：“移动互联网和传统互联网的测试有什么不同”，这个问题太大，不好回答。简单地说，我觉得从测试用例的设计方法来讲没有什么特别，可能更明显的区别在于移动端系统和业务的特殊性。随着经验的积累，大家会发现移动端在设计用例的时候需要考虑的点非常多，非常杂。所以设计出好的用例必须对应用以及对应的系统要有足够的认识才行。在本章中作者根据自己的经验列举了一些相关的要点，但绝不仅限于此。

4.1 多分辨率测试

从目前来看 Android 智能机是以支持多分辨率的移动端设备著称的。从发展历史上来看，其支持的设备几乎涵盖了各种分辨率：小到一只手能握住，大到打电话的时候需要整个脸贴上去，应有尽有。我们看一下 Android 官方文档就知道能支持多少种分辨率了。

	Low density (120), *ldpi*	Medium density (160), *mdpi*	High density (240), *hdpi*	Extra high density (320), *xhdpi*
Small screen	**QVGA (240x320)**		480x640	
Normal screen	**WQVGA400 (240x400)** **WQVGA432 (240x432)**	**HVGA (320x480)**	**WVGA800 (480x800)** **WVGA854 (480x854)** 600x1024	640x960
Large screen	**WVGA800** (480x800)** **WVGA854** (480x854)**	**WVGA800* (480x800)** **WVGA854* (480x854)** 600x1024		
Extra Large screen	1024x600	**WXGA (1280x800)**† 1024x768 1280x768	1536x1152 1920x1152 1920x1200	2048x1536 2560x1536 2560x1600

对多分辨率的测试可以考虑使用模拟器（Emulator）和真机（Device）。由于各个运营商在智能机 Rom 上都稍有改动，所以建议尽量在真机上进行应用的测试。如果没有真机而需要使用模拟器的话，其测试结果仅供参考。在模拟器上主要是进行应用的界面和功能测试，两者的测试结果都无法和真机相提并论。下图是目前 Android 官方统计的各种分辨率的使用情况。



	ldpi	mdpi	tvdpi	hdpi	xhdpi	xxhdpi	Total
Small	9.8%						9.8%
Normal	0.1%	16.1%		37.3%	24.9%	1.3%	79.7%
Large	0.6%	2.9%	1.0%	0.4%	0.7%		5.6%
Xlarge	0.2%	4.5%		0.1%	0.1%		4.9%
Total	10.7%	23.5%	1.0%	37.8%	25.7%	1.3%	

iOS 设备支持的分辨率主要由几个固定的机型来主导，iPhone/itouch、ipad、iPhone5 3 者，分别有普通屏幕分辨率和 Retina 屏幕分辨率。从支持的分辨率数量上来看，的确比 Android 系统好很多。但是由于 apple 设备普遍比较贵，所以很多企业也不是所有类型的设备都配备。在这里需要提到的是，Xcode 自带的模拟器不支持一些越狱产品的测试，所以在使用上很受限制。一般情况下，对 iOS 应用的测试都是使用真机进行测试。下图是截止 2013 年 4 月 iOS 的分辨率分布统计（来自友盟）。



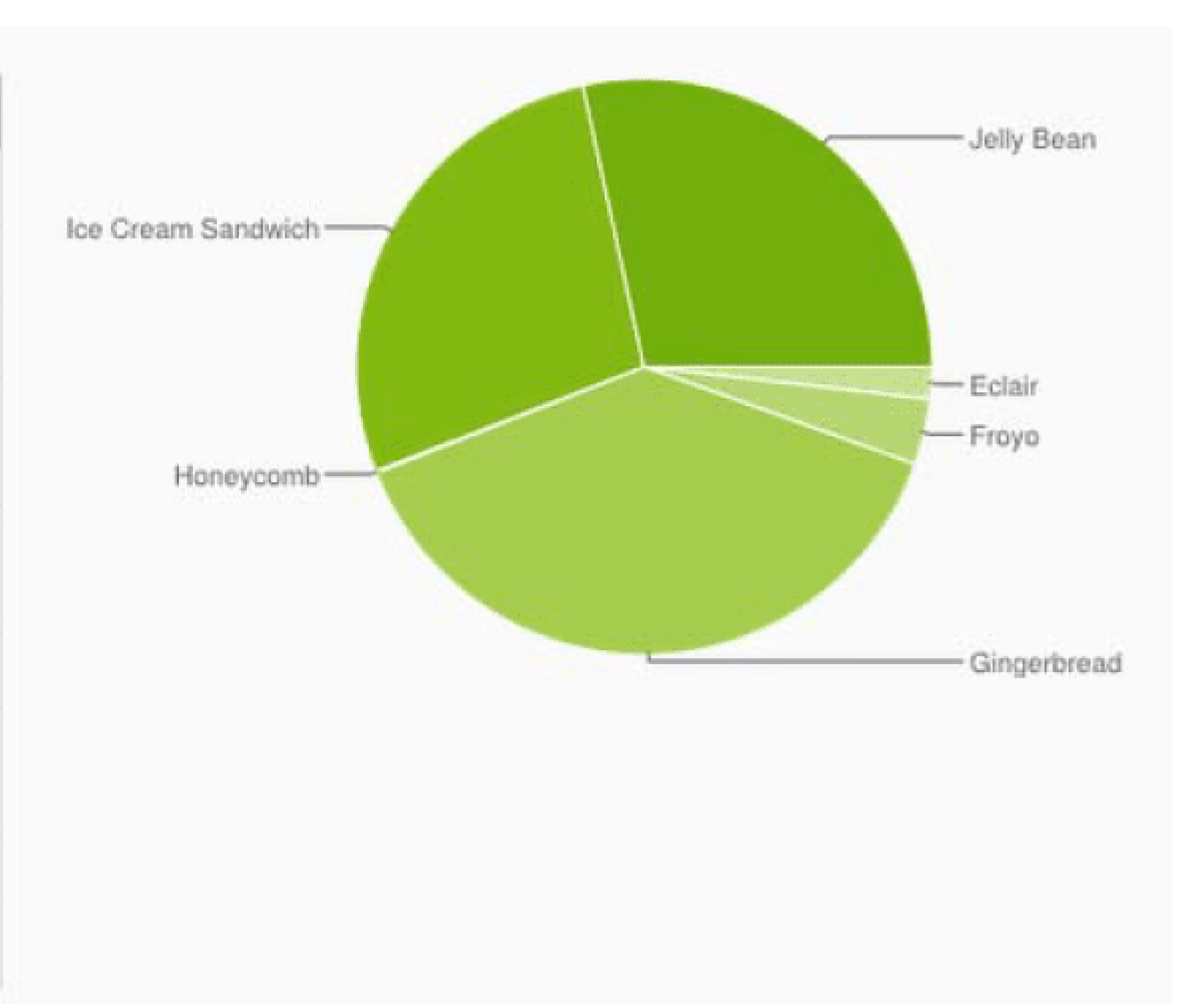
排名	分辨率	占有率
1	640*960	56.6%
2	640*1136	17.7%
3	768*1024	14.4%
4	1536*2048	9.4%
5	320*480	1.9%

总体来讲，在模拟器和真机上进行分辨率的测试是很有必要的，毕竟是用户视觉最直接的感受，因此是很重要的测试点。

4.2　多系统测试

继多分辨率测试之后紧接着的就是对多系统（Rom）支持的测试。先看一下目前 Android 系统不同版本在市场上的活跃度分布情况吧。

Version	Codename	API	Distribution
1.6	Donut	4	0.1%
2.1	Eclair	7	1.7%
2.2	Froyo	8	3.7%
2.3 - 2.3.2	Gingerbread	9	0.1%
2.3.3 - 2.3.7		10	38.4%
3.2	Honeycomb	13	0.1%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	27.5%
4.1.x	Jelly Bean	16	26.1%
4.2.x		17	2.3%

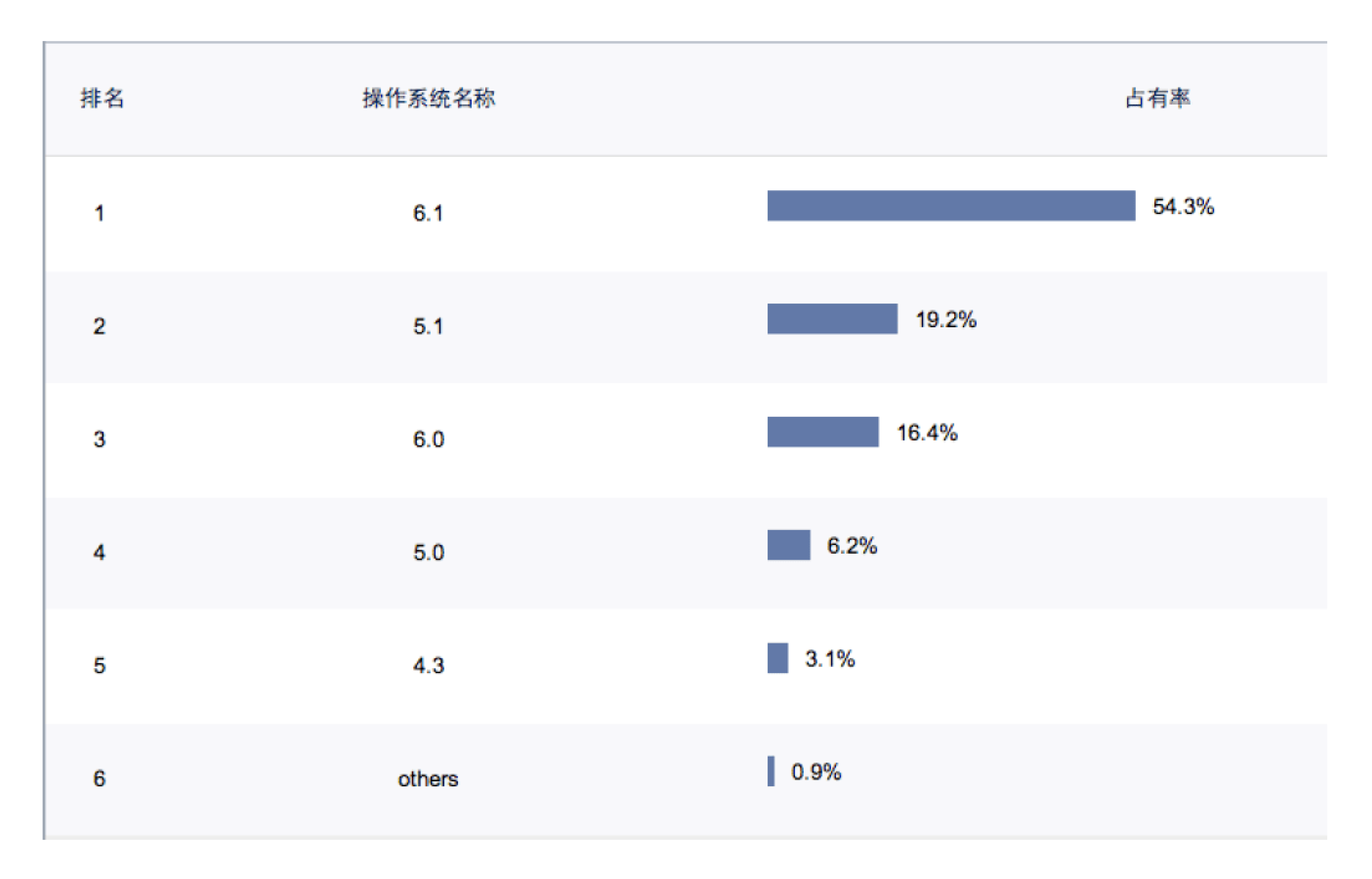


不难看出，目前大部分应用是支持 Android2.3.3 及以上的版本系统的。Android 最新推出的 4.×.×也已经有了很可观的用户量。对多系统的测试比对多分辨率支持的测试还要麻烦，现在各种官方修改系统和自制系统层出不穷，例如小米、魅族、锤子、CM 等等。这类系统除了界面改变很大之外，部分定制化的系统甚至会改变系统内的一些接口，从而导致应用功能无效或者崩溃。

使用 iOS 系统相对好点，但是由于 apple 服务器的验证机制原因，所以 iOS 的系统只能单向进行升级，不能降级。



提示：这里推荐一个小工具——TinyUmbrella，这个工具能够备份以前 iOS 系统的 shsh，在使用 itunes 升级的时候欺骗 apple 的服务器，从而达到降级的目的，其官方网站为：http://thefirmwareumbrella.blogspot.com/，不过貌似该项服务已经被禁止，有兴趣的读者可以去尝试一下。下图是截止到 2013 年 4 月的 iOS 各个版本的系统使用分布（来自友盟）情况。



排名	操作系统名称	占有率
1	6.1	54.3%
2	5.1	19.2%
3	6.0	16.4%
4	5.0	6.2%
5	4.3	3.1%
6	others	0.9%

从上图中的数据可以得知，apple 的用户群大多都有升级到最新系统的习惯，虽然他们大部分不关心到底系统升级了什么功能，但对测试工程师而言，我们需要尽量确保能够在 5.0 以上的所有系统中测试自己的应用。



提示：另外需要特别提醒的一点是，我们在测试的过程中需要注意 iOS 系统每次升级后，是否有一些新的资源库支持。在新平台开发应用过程中，很可能会用到一些新的资源库，这些资源库在最新的 iOS 系统中能够更简便地支持一些功能，但是也可能完全不兼容低版本的 iOS 系统，导致的结果就是，一旦应用在低版本系统中调用了该库的功能，应用将直接闪退（Crash）。

4.3 用户不同的使用习惯

4.3.1 Android 权限问题

Android 系统和 iOS 系统在市场上都有着自己系统的特殊性。这些特殊性被各种应用以及广大的用户在不知不觉中使用着，同时也对第三方应用产生非常大的影响。

在 Android 系统下的项目工程中往往需要为应用获取各种权限，比如，需要通过短信验证的应用需要有短信（SMS）的使用权限，智能拨号联系人需要有系统联系人（Contacts）的权限使用，拍照美图软件需要有照相机（camera）的使用权限等。添加权限需要在项目工程中的 AndroidManifest.xml 中进行设置，添加类似：

<uses-permission android:name="android.permission.CAMERA"/>字符串即可。

现在安全软件越来越多，比如 360 卫士、91 助手等等。这些软件都提供了告知用户自己安装的软件到底使用了哪些权限的功能，这使得应用偷偷使用权限之事对所有的小白用户都变得透明了。Android 市场中有着太多的应用，在其被使用的时候，程序会在后台悄悄地获取用户的数据，更有甚者偷偷地下载垃圾软件或病毒并把它们安装在用户的机器上。自然地，这也使得很多以前使用正常的功能变得让用户难以接受。比如某联系人应用获取了如下的权限：

```
<uses-permission android:name="android.permission.CAMERA" />
<uses-permission android:name="android.permission.FLASHLIGHT" />
<uses-permission android:name="android.permission.VIBRATE" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<uses-permission android:name="android.permission.CHANGE_WIFI_STATE" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="com.android.launcher.permission.INSTALL_SHORTCUT" />
<uses-permission android:name="com.android.launcher.permission.UNINSTALL_SHORTCUT" />
<uses-permission android:name="android.permission.BAIDU_LOCATION_SERVICE" />
 <permission android:name="android.permission.BAIDU_LOCATION_SERVICE" />
<uses-permission android:name="android.permission.NFC"/>
<uses-permission android:name="android.permission.READ_SMS"/>
```

从图中可以看到大部分权限的使用还是比较中规中矩的，但是其中的两项：NFC 和 READ_SMS 权限的使用就会让用户产生困惑。作为测试工程师，需要了解被测对象使用了哪些权限，为什么使用，是否会对用户造成困扰等问题。

4.3.2　Android 硬件问题

Android 智能机系统层出不穷（这点在之前已经提到过），同时硬件差异也很大。硬件上的差异同样会造成软件上的缺陷，甚至是恶劣的体验。有的机器拥有硬键盘，这对部分依赖虚拟键盘进行输入的软件是极大的挑战，开发和测试工程师需要有针对性地适配和测试，否则会造成应用的界面甚至功能上的缺陷。



有的智能机拥有实体方向键或者滚轮，这会使软件在使用过程中的焦点发生移动，变得异常诡异，稍不小心就会造成空指针缺陷（Null Pointer Exception）。如下图的机器。



此外还有一部分智能机支持双卡双待，大部分智能拨号软件或需要读取 Sim 卡信息的软件在这类机器上会甘拜下风。这类手机在山寨市场中有很大的用户群体，所以测试工程师依然不能对它们放松警惕。



随着 Android 的发展，系统在软件和硬件上都得到了很好的提升。目前带有滚轮和硬键盘等的机器已渐渐退出市场，取而代之的是各种大屏幕、高分辨率的机器。在这类机器上，一部分应用还是会持续受到硬件（比如摄像头、GPU）和 Android 自制系统带来的折磨，这对广大企业仍然是一个巨大的挑战。

4.3.3　Android 操作习惯

说完权限和硬件，接下来谈一下用户在 Android 系统上特有的操作习惯。基本上有以下几类：

- 菜单键（Menu）
- Home 键
- 返回键（Back）
- Home 键长按（现在最新的 Android 系统已有单独的控件按钮代替了这个功能）
- 显示当前进程列表（同 Home 键长按）
- 调整音量
- 待机

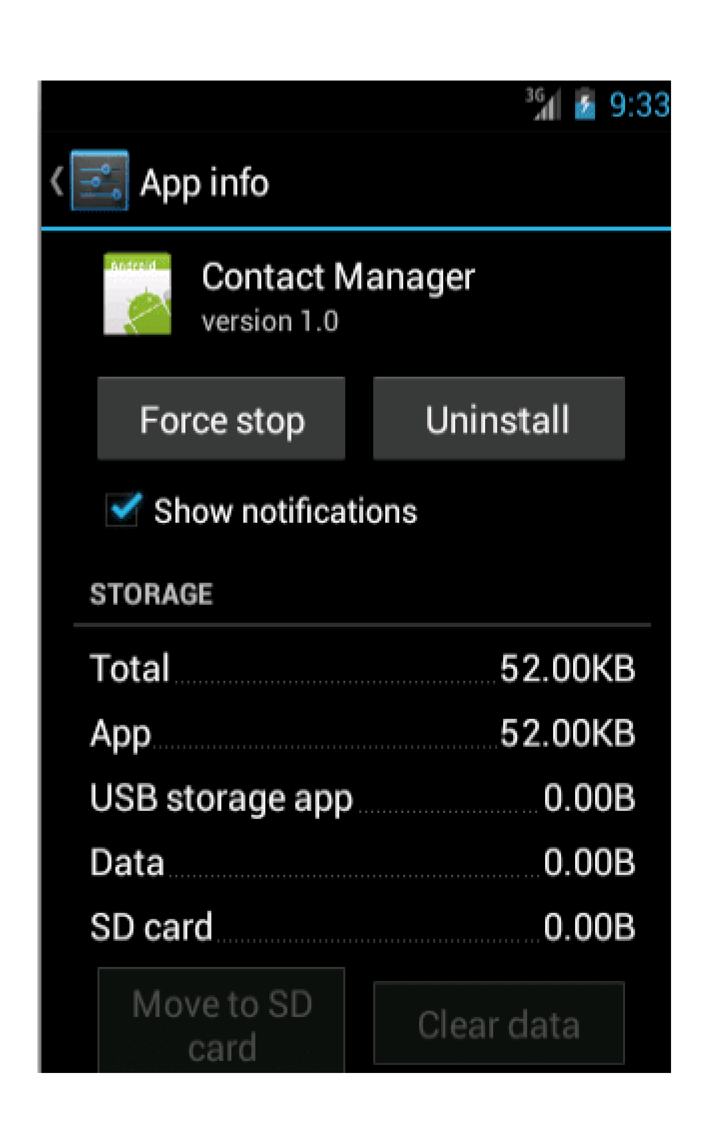
其中用户最常用的是菜单键、Home 键、待机和进程列表这几个按钮。相应的，作为应用测试工程师我们需要考虑的项就变成了：

- **应用**中的 Back 键的事件是否重写了？在各个功能界面中点击 Back 键会有什么反馈?
- 用户在**应用**各个界面点击 Home 键之后，再次打开**应用**的反馈：应用是默认处于后台的状态还是有结束生命周期。
- **应用**是否使用到菜单键，切不可遗漏这个按键和**应用**的交互。
- **应用**在任何状态下，系统进入待机或者关机。当系统被重新唤醒后，**应用**处于什么状态，是否有正确的反馈。
- **应用**进程被用户手动关闭之后，是否可以再次正确启动。
- ……

功能测试不仅限于以上这些测试点，不过用户常见的这些习惯需要时刻贯彻在测试过程中。有一部分测试工程师过于专注在自动化测试和业务测试上，而忽略掉真正的用户是怎么使用产品这个很重要的测试点。

4.3.4 Android 数据的移动或清空

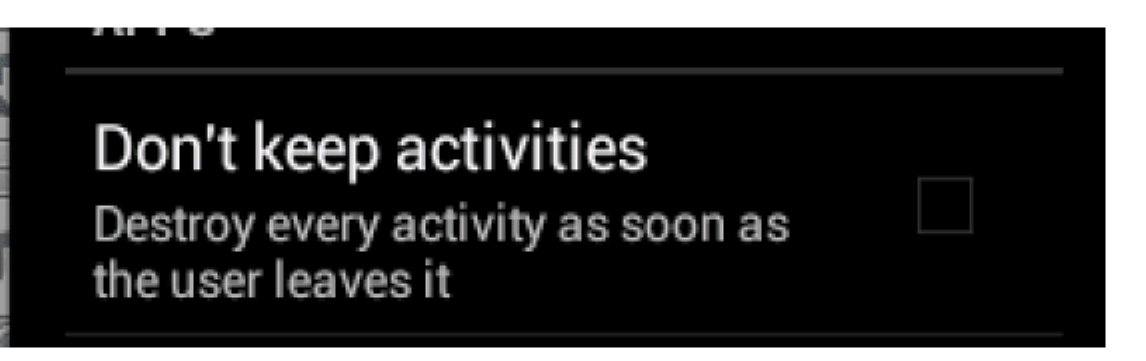
Android 智能机用户除了一直需要充电、喜欢刷系统、接触各种山寨应用商店之外，还有一个常见的习惯——移动或清空应用数据，其操作界面如下。



大部分使用 Android 系统的中低端机器的硬盘和内存容量都不大，所以 Android 用户很在乎应用（apk）的大小。用户会在软件没有响应或者出现重大缺陷的时候，点击强制停止按钮（我们当然是不希望他点卸载按钮的），而大部分 Android 的游戏支持将数据转移到 SD 卡中运行，因此用户也在一些应用数据很大的时候选择清空数据来释放自己机器的空间。这些都是我们测试过程中需要去验证的，不能忽略。

Android 系统中还有一个比较恶心的问题。某些第三方软件会提供给用户一些选项，意思是能够自动关闭在后台运行的程序。如果仅仅是关闭后台运行的程序，我觉得对用户来讲是件好事。但可恨的是这个选项其实大多是勾选了 Android 开发者设置中的一个

销毁（Activity）选项。



从对这个选项的描述可以看出，它与“关闭后台程序”这个说法还是有一定差距的。该选项对应用的杀伤力非常巨大，了解 Activity 的工程师肯定心知肚明。一款应用中，如果有某项功能需要保留一个 Activity 状态，而用户又选择了这个选项，那么该项功能就会全部失效。这当然不是应用自身的缺陷，但需要提醒广大工程师注意有这样一个“坑”。

4.3.5　iOS 操作习惯

Android 系统的这些特性就说完了，回来咱们再说说 iOS 系统。iOS 系统与 Android 系统相比，无论是碎片化问题还是安全问题都解决得好太多了，不过同样也存在着一些令人很头疼的问题。

首先是用户操作习惯。由于 iOS 系统的特殊性，所以用户最常用的操作还是比较简单的：

- 单击 Home 键
- 双击 Home 键
- 关闭当前某应用进程
- 打开或关闭音量
- 调整音量
- 待机

与前面提到的 Android 系统一样，测试工程师需要时刻记住用户的这些操作习惯，

清楚地了解每个操作对系统应用生命周期的影响。下图是 iOS7 以下系统和 iOS7 的双击 Home 键截图。





4.3.6 iOS 越狱问题

接下来讲讲在 iOS 系统中用户最最常用，企业最最头疼，但好像又没有多少测试工程师关心的问题——越狱。

由于 iOS 系统的严密性导致很多好的应用不能直接安装，必须花钱买才能使用，于是免费安装收费应用就变成了广大群众的刚需。有了需求必然会有回应，系统越狱工具就这样诞生了。我们先来看下市场上的常见的几个越狱工具：红雪、绿毒和 jailbreakme。







这里需要测试工程师注意的是，实践经验告诉我们，在相同的 iOS 环境下使用不同的越狱工具，一些应用的部分功能所表现出的现象可能不一样。当然，这样的问题毕竟只占少数，不是重点，重点是越狱之后用户做了些什么事情，下面试举一些：

- 开始安装各种收费的游戏。
- 开始安装各种 Cydia 里面的插件。
- 开始安装各种垃圾流氓软件。
- 开始拿 iOS 设备当 U 盘使用。
- ……

还有很多，只有想不到，没有做不到。若说乔布斯是被这部分用户气死的也不为过。



提示：什么？只有中国用户是这样的？那是有点不好意思，不过，毕竟我们面对的大部分是中国用户，更何况国外也不是没有这样的用户。但是，企业的开发和测试工程师对越狱这码事可就头大了。

因为越狱之后，最常见的一个问题就是系统变得不稳定，应用之间发生冲突。简单举几个例子。

1. 输入法

成功越狱之后，相信很多用户最先做的就是安装自己喜欢的输入法，但是这些输入法往往就是导致很多其他应用崩溃的罪魁祸首。在这点上测试工程师需要特别注意，要在应用的各种输入框内尝试使用不同的输入法，避免疏漏。否则，对用户来讲，他们只会不买这款崩溃的应用的帐，而不会不买输入法的帐。

2. 美化主题类的插件

如果单单美化主题是没有什么影响的，可一部分美化插件不仅会让其他应用界面的皮肤变得不堪入目，还会造成自己崩溃。同样，用户不会认为这是美化主题的错，用户永远是爱美的。

3. 垃圾插件或手机管理软件

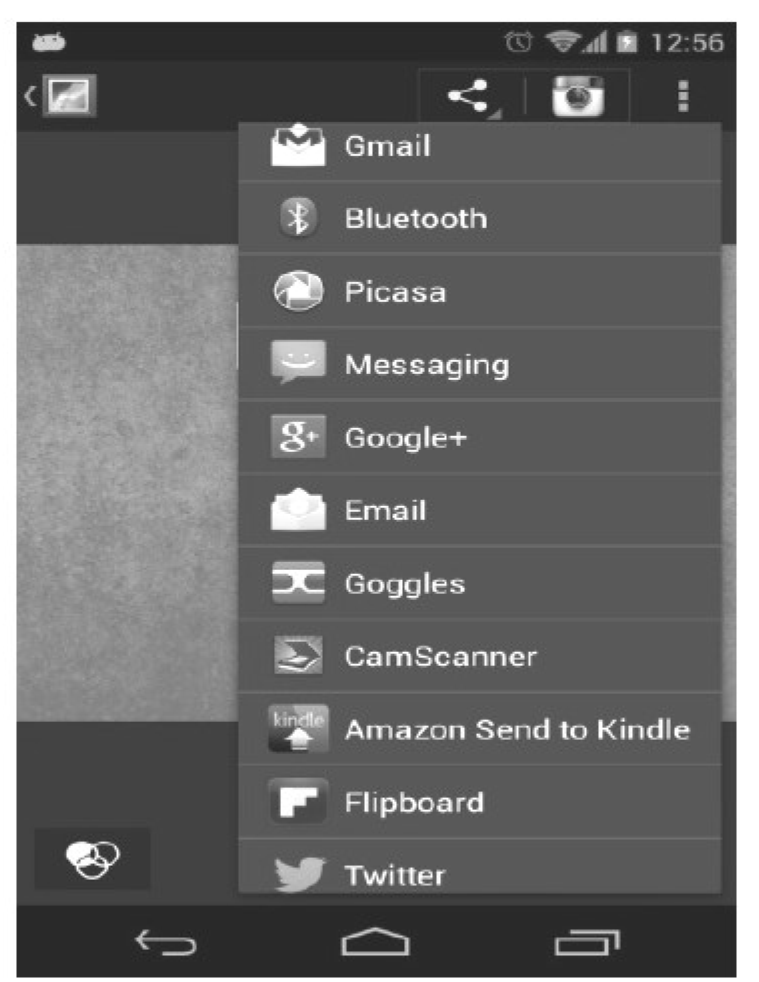
这里就不特别指出某些应用了。一旦越狱之后，iOS 系统马上就变得和 Android 系统一样开放。现在很多第三方应用的登录方式都集成使用了新浪微博、腾讯微博、豆瓣等帐号。之后陆续也推出了采用 SSO 模式的（Single Sign On，具体可查看 http://baike.baidu.cn/view/190743.htm）登录方式，更方便了用户使用社交网站帐号进行登录。问题是，用户越狱之后某些管理软件会将后台应用强制关闭，这样直接就导一致用户无法使用 SSO 方式进行登录。现象就是死循环——从应用 A 跳转到社交应用 B →选择自己的帐号，正常情况下会直接使用该帐号登录，可现在却变成了跳回 A 继续选择社交应用，如此往复。

总体来讲，越狱对用户还是有很多好处的，但对应用开发者来说就是一场噩梦。问题会变得非常复杂并且难以定位。无论是 Android 系统还是 iOS 系统，测试工程师都需要关注真正的用户是怎么使用移动设备的，而非仅仅关注日常业务和眼前的自动化测试。

4.4 网络的不稳定性

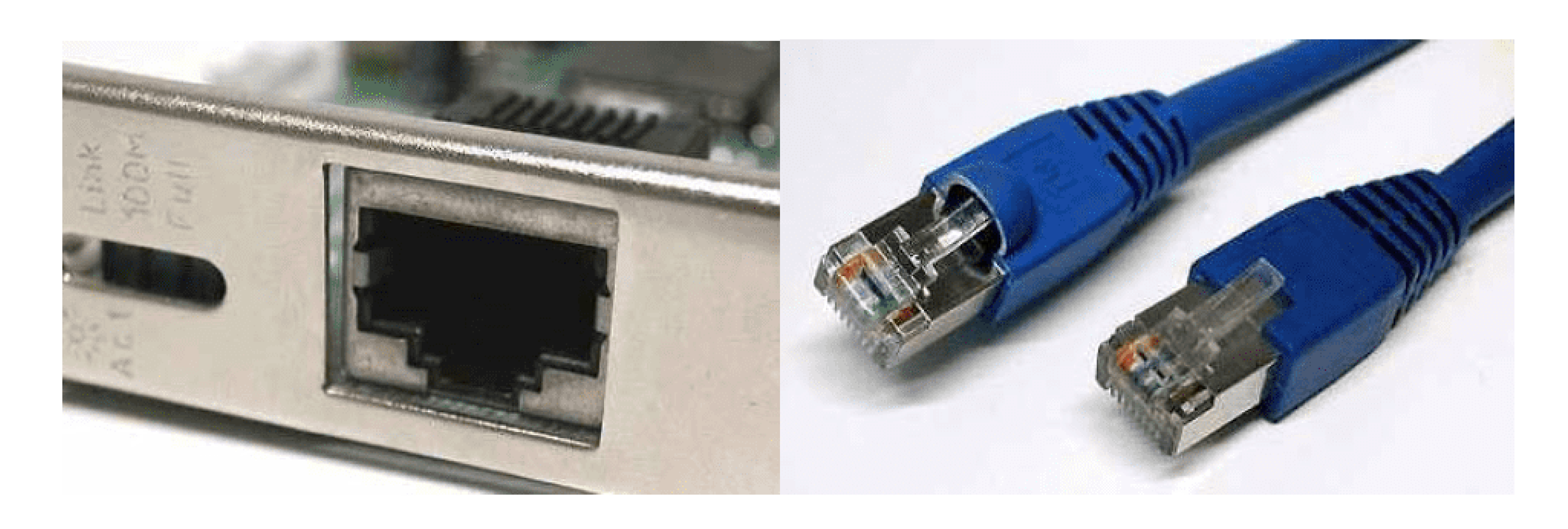
移动互联网之所以比传统互联网更具有传播力，流行得更快，正因为人们可以随时随地拿出移动设备来上网，发微博、做分享等等。从 2G 网络到 3G 网络的快速转变可以看出用户对网络的依赖性非常强。随着应用的界面越做越精致，浏览的信息量越来越大，网络变成了生活中不可缺少的元素。人们开始习惯每天使用移动设备浏览大量的信息、分享所见所闻，甚至游玩网络游戏。


中国移动
15:20
92%
Environment Special: The Perils of Plastic
ZAKER
TIME
Inside
Steve's
Pad
How Jobs works
The tale of the tablet
新闻头条
微博广场
互联网新闻
新浪微博
TechWeb
南方周末
应用推荐


12:56
Gmail
Bluetooth
Picasa
Messaging
Google+
Email
Goggles
CamScanner
Amazon Send to Kindle
Flipboard
Twitter


CLASH of CLANS

事实上，人们的行为习惯并没有改变，而是行为模式因移动互联网的出现而得到改变。在移动互联网中，网络不像传统互联网那么稳定，传统互联网基本上都是通过实体接口连接网线或长期在稳定的无线网络下使用。



移动设备大部分时间更依赖于 2G、3G 等网络。可是在城市各个地区总有一些网络信号不好或覆盖不到的区域，比如地铁、电梯、建筑物内部等。



就用户而言，他们只有在无聊、空闲下来的时候才会拿出移动设备进行操作，而此时所在的地点一般就是地铁、电梯或洗手间等。很不幸，这些场所的网络基本上信号强度都比较差。在如此“艰苦”的环境下，一款应用要对网络连接变化，容错等做大量的测试就显得尤其重要。

对移动设备而言，连接不上网络是用户无法忍受的，面对这些“蛮横”的用户，应用需要给出相对正确的提示。这个问题在第 3 章用户体验中有提到过，并介绍了若干例子。因此我们在测试的过程中，需要将网络连接不上、无网络、企业服务器错误等各种情况予以考虑并进行细化的测试。现在大部分的应用是，无论发生什么情况，全部提示用户“网络错误，请稍后再试”。从我个人角度来看，这是对用户不负责的一种做法，作为测试工程师需要完完全全地站在用户角度去思考问题。针对网络连接我们提出以下一些建议：

网络不稳定，没有得到服务器的反馈

应用可以提示用户“网络不稳定，请稍后再试”。

网络没有连接

应用可以提示用户“网络目前没有连接，请重新连接网络之后再尝试”。

企业服务器或者第三方服务因未知原因出错

应用可以提示用户“目前无法接收相关数据，我们正在调查，请稍后再试”。

……

对于网络，除了网络通畅和无网络这两种极端情况之外，大多数时候处于中间状态。不同网络之间的切换也需要进行模拟测试。从技术上来讲，模拟器或者某些其他的工具可以模拟相关场景，但就移动互联网应用的网络状态切换而言，我更建议进行实际场景的测试。

举例来说，用户最常碰见到的一个场景，是原本非常畅通的 3G 信号突然变弱，然后有一断开瞬间，再重新连接到网络上。从网络本身的变化而言，这个过程非常复杂，但用户看到的就只是 3G 信号变弱，然后断开，马上又连接上了。对于应用，断开的过

程中可能会用到一个连接超时设置，会有多次请求的策略等，只不过这些用户都不知道也无须知道而已。那么，这种情况应该如何测试呢？与其设计复杂的方法进行模拟，还不如直接花一天时间，带上必需的设备，进入那些网络信号很差的实际场所测试。这样既不浪费时间，还能达到最接近用户使用场景的目的。

这里还需要提到的一点是，一些在线及时通讯的应用或游戏都会使用长连接的方式来达到让应用更稳定的目的。当被测应用有长连接支持的话，那么除了上面所说的容错和网络切换测试之外，还需要在不同的网络环境下，根据服务器设置的心跳时间来进行边界值的一些测试，这也是极其重要的。

4.5 安装/卸载测试

可能很多人觉得安装卸载测试在任何一种测试中都是重点，没有必要在移动应用的功能测试中特别提到。那么为什么还要提到呢？因为“坑”实在太多，而且一旦出错就是无法启动的优先级最高的错误。作为测试工程师，这个测试点是必须重视的测试点，同时也是一个需要深入探究的测试点。

在 Android 和 iOS 系统中，安装方式有多种，应用是否能够经得住考验是测试工程师需要关心的。要考虑不同途径的安装之后，应用是否可以正常启动？功能点是否不会受到影响？等等。

先来介绍使用命令行方式安装应用。什么？用户根本不会用？首先我想说你太小看 Android 用户的动手能力，其次命令行安装原本就是从业人员的基本技能，安装之后必须要能够正常使用。使用 adb install <package name>进行安装，安装成功后会看到如下图样式显示的内容。iOS 的用户则一般使用命令行进行应用的安装。

```
37 KB/s (25921 bytes in 0.673s)
        pkg: /data/local/tmp/ContactManager.apk
Success
applematoMacBook-Pro:ContactManager apple$
```

再说说第三方软件的安装。这也是目前大部分 Android 小白用户最喜欢使用的方式。大部分的 Android 用户和 iOS 越狱用户会安装豌豆夹、91 助手等软件进行傻瓜式的安装和管理。



使用第三方软件进行安装，从逻辑上来讲应该和命令行安装没有区别，但是由于现在市场上这类渠道和软件实在太多，测试工程师需要进行实际的操作验证之后，才能保证这个测试点可以通过。

一部分用户或者平台会先将应用下载到 SD 卡中再进行安装（这也是测试需要注意的点），有些 Android 应用由于加载数据量太大，所以允许用户将应用的部分数据在智能机内存和 SD 卡中互相转移。数据互相转移之后应用需要能够正常运行，这也是常常会被遗漏的测试点。



还有一部分数据量大的 Android 应用（一般是游戏）会采用小体积的应用安装，之后通过在线下载数据的方式使用。由于 Android 用户的内存容量与 iOS 相比用户要宝贵得多，所以这类方式更容易让用户接受（当然，用户下载的时候并不知道还需要后继地在线下载数据，也是一个下载陷阱吧）。应用本身安装之后，重点还需要查看以下几点：

- 不联网的情况下是否能正常运行，会不会崩溃？
- 在线下载数据不完整，能否再次启动？
- 是否支持断点续传？
- 下载完整之后，再次启动。
- 安装成功后，检查版本号以及相关数据。

从在线下载数据的应用来看，下载过程也是安装的一部分，但测试的复杂程度比一般应用要大的多。

总体而言，移动互联网中由于 Android 系统的开放程度太高，导致其复杂程度比 iOS 系统要大很多。单纯从安装卸载来看，Android 有各种工具、平台、渠道包等，五花八门，而 iOS 就控制得很好，只有 itunes、app store 和 testflight 下载。

说到安装测试就不得不提升级测试，升级测试在移动互联网中也是非常重要的一个测试点，升级功能测试不全面等同于给一个人断粮断水，稍不小心就会葬送一个应用的“生命”，因而要额外注意。下一节我们就来详细聊一下升级测试。

4.6　升级测试

如果用户会长时间地使用一款软件的话，那么一定要让用户安全升级。用户升级之后，也许会因为用户体验方面的原因给应用一个差评，就如前不久新版手机 QQ 客户端升级之后，广大用户群众只给了 1 星。企业当然都是希望应用的每次更新都能够留住老用户，同时带来更多的新用户，不过“理想是美好的，现实是残酷的”。不过无论是

美好的理想还是残酷的现实，前提是用户能够正常地进行覆盖安装。

上一节介绍过的安装方式都能用于进行应用的升级，所以本节就仅关注“升级”这个功能点。

一个应用可以被升级有几个必要条件：

- 旧版本的应用和新版本的应用拥有相同的签名
- 旧版本的应用和新版本的应用拥有相同的包名
- 旧版本的应用和新版本的应用需要有一个标示符来区分（一般使用应用版本号做为标示符）

这些条件在 Android 和 iOS 系统中大部分相似，这里就不做详细说明。下面主要来说一下在 Android 系统中进行升级安装的常见方式。

4.6.1　增量升级

增量升级也叫做差分升级（Smart App update）。简单来讲，假设一个旧版本 Apk 是 5M，新版本 Apk 是 8M。那么传统的升级是完全下载 8M 的应用之后，对 5M 的应用进行覆盖安装。如果使用增量升级的话，需要下载更新的部分可能只有 3M，当然，差分包的大小差肯定不是做简单的加减法。但是，对用户而言，这种方法有个明显的好处就是不必再下载一个完整的 Apk 进行安装。

增量升级也有明显缺点：

- 如果一款应用有多个版本，并且每个版本都有用户在使用，那么增量升级就变得非常麻烦。虽然差分的补丁（Patch）容量是小，但是只能针对单一的旧版本应用。也就是说，必须对发布的所有旧版本和最新版本做差分。虽然补丁可以由脚本自动生成，但是依然很麻烦。
- 如果一部分系统没有拿到内置应用的权限，那么就没有办法进行增量升级。

直到现在，我依然没有看到增量升级有比较广泛的运用，也许就是因为有这样、那样的不稳定因素所致吧。

4.6.2 内置应用升级

部分应用内置在系统（Rom）内部之后，企业也会希望应用支持更新升级到最新的版本。由于 Android 系统版本繁多，内置以后的情况也变得复杂，需要具体情况具体分析。如果应用中有.so 文件的话（比如输入法），往往内置系统之后，应用的.so 文件一般会保存在/system 级的目录下，也就是说，无论用户还是应用都没有访问的权限。但应用的其余一些资源文件却可以进行升级。那么测试工程师就需要注意每次升级之后的应用是否与内置在系统中的.so 文件匹配，如不匹配则会直接导致应用崩溃。



提示： 升级测试场景可能还有很多，不过我的原则是没有实践过就不会写出来。所以只好等我以后再做补充了。应用升级并非仅仅检查新功能或做冒烟测试，更多的是需要去了解升级的原理，升级到底更改了哪些文件，而这些文件又影响了哪些功能和数据库。找到这些问题的答案才能有针对性地进行全面的测试。

4.7 并发测试

移动设备测试和传统测试区别还于，移动设备的使用过程中并发情况较多。这里说的并发不是指服务器性能测试中提到的并发，而是由于智能终端是一台多功能机器，会有多个应用同时启动或多个事件同时触发的情况。测试工程师应该有独立的测试用例对应这类情况。这里主要列出一些常见的并发情况。

4.7.1 弹出框提示

比如闹钟的提示、低电量、短信等的提示。无论在何种应用中，这类提示都会出现。

此类提示会让目前正在运行的应用进入暂停（Pause）状态，待用户响应操作完毕之后才会继续运行。测试工程师需要关注应用是否能够在暂停之后正常继续运行。





4.7.2　另一个应用启动

比如正在使用一个应用，此时有一个来电，或是用户按下快捷键启动相机等诸如此类的情况。出现这类情况对于智能机来讲再普通不过了。一般这类场景需要特别注意应用几种常见的特殊状态：

- 应用正在播放视频。
- 应用正在向服务器连接发出或接受请求。
- 应用正在下载数据或升级。
- 用户正在进行输入。

应用处于上述这些状态时，很容易出现不可预见的问题，需要特别注意。

4.7.3 关机或待机

相信这个测试点不需要多说明，用户日常最常见的操作即是待机和关机，测试工程师不但需要关注在恢复正常后应用的功能是否正常，还需要关注应用数据是否会因为关机和待机而丢失。

4.7.4 功能冲突

最常见的功能冲突就是音乐和语音功能冲突。现在越来越多的应用开始有了语音留言、录音、录制视频等功能。当正在播放音乐或视频的时候，使用应用的这些功能必须强制将现在播放的视频或音频暂停。对这类功能的测试不仅仅需要确保应用自身的正确性，还需要考虑到其他被暂停的应用是否能够正常继续工作。

4.7.5 可存储设备

许多应用的功能会涉及到读取系统中的一些资源。在正常使用的情况下，用户获取资源（比如智能机中的音乐或图片）是不会出现问题的。但是当把智能机当作移动存储设备时，再使用应用的这些功能的话，就很容易导致应用的崩溃。

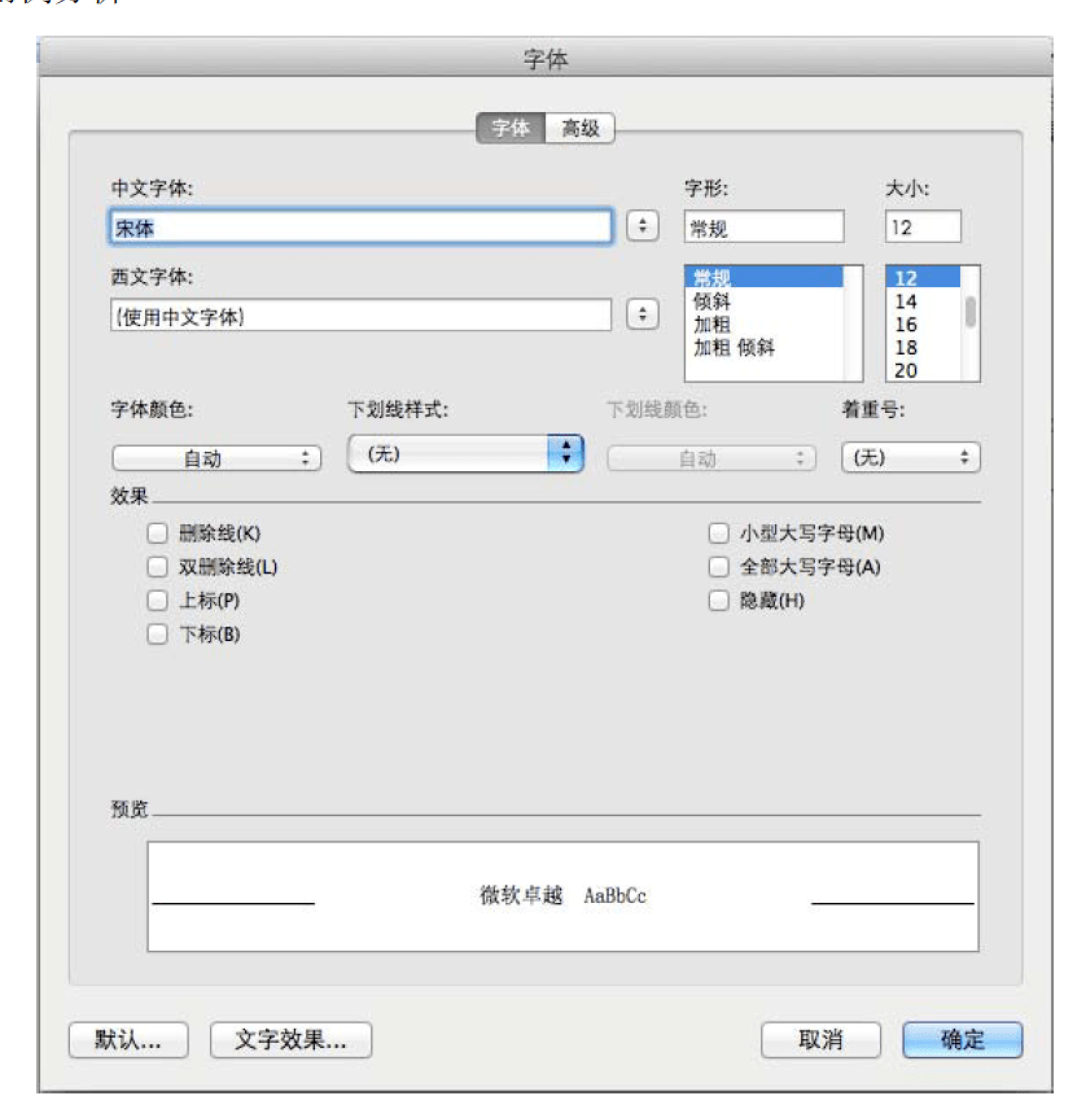
由于将来的智能机功能会越来越多，在智能机上出现的并发事件就会更频繁。对于测试工程师而言，重现这些并发事件不是一件容易的事情，这不仅仅需要技术，更需要对于系统、用户习惯有更多的了解。

4.8 数据来源

在移动互联网中，用户开始慢慢习惯将数据存放到云端，因此也出现了很多相关的服务应用，比如印象笔记、dropbox 及各种网盘等。在许多移动互联网应用的测试中会

和这些数据打交道，而这些数据在移动领域中存在着非常大的“坑”。

我们先来看一个在传统桌面系统上很经典的测试用例分析题，对如下图所示的界面做测试用例分析。



该对话框中有很多输入框需要测试，在输入框中输入字符串主要有以下 3 种形式：

- 直接输入。
- 选择软件提供的选项。
- 从粘贴板中复制粘贴获取。

也许现在的技术也支持语音输入，应该算是第 4 种了。这个例子充分表明，很多情

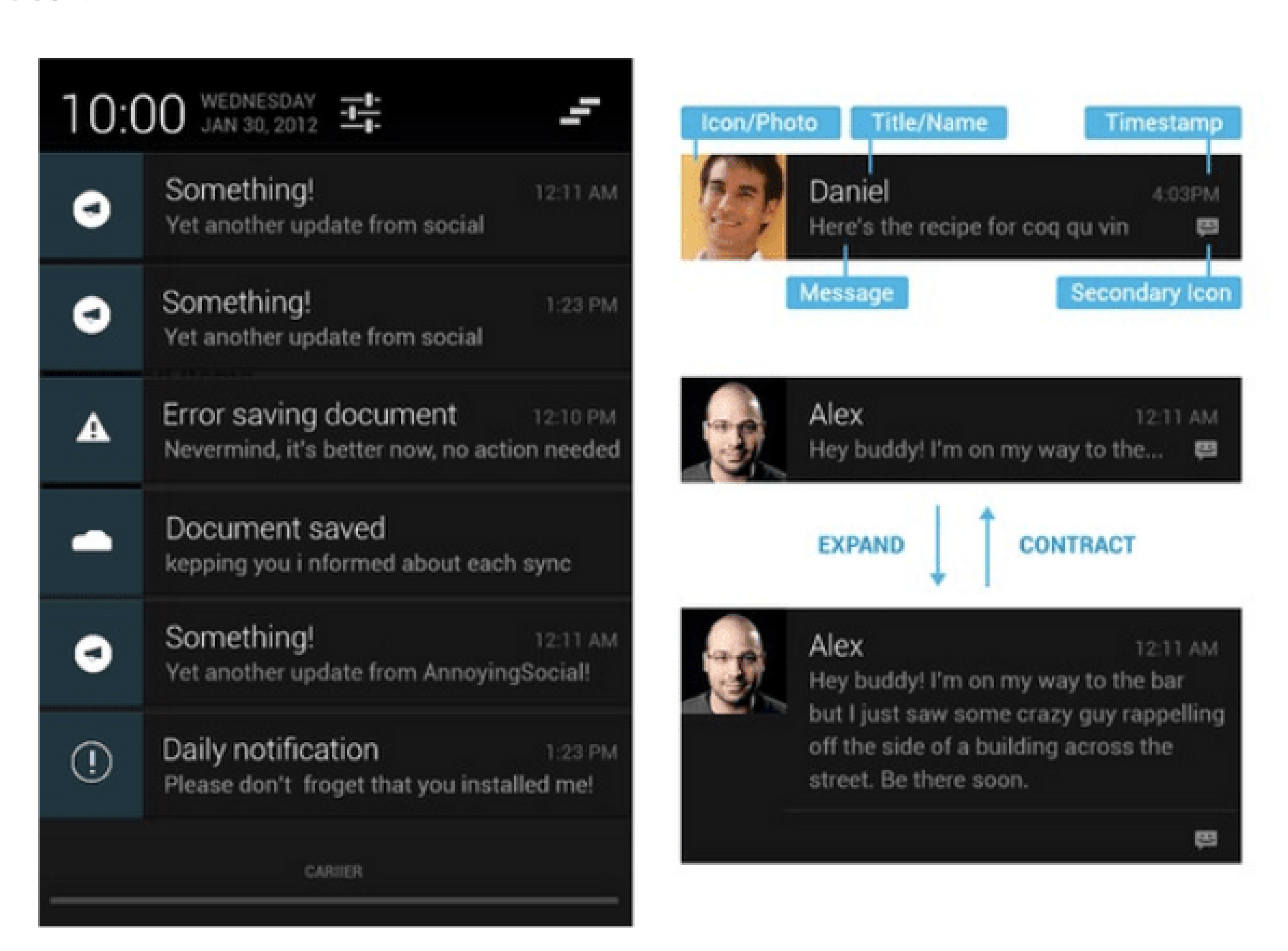
况下，数据并非只通过单一途径才能够获取，在移动互联网应用中更是如此。我们在测试的时候，往往想到的场景和文件来源，仅仅是满足功能设计需求文档的要求，而用户获取文件的方式可谓是千奇百怪。在移动互联网应用的测试过程中，如碰到与文件相关的测试，要想使测试用例覆盖得尽量全面，那就必须在日常生活中使用自己的产品，要经常和使用产品的用户进行沟通，把自己放在用户的角度去思考才有可能。关于本节的扩展内容会在第 7 章中详细叙述。

4.9 推　送

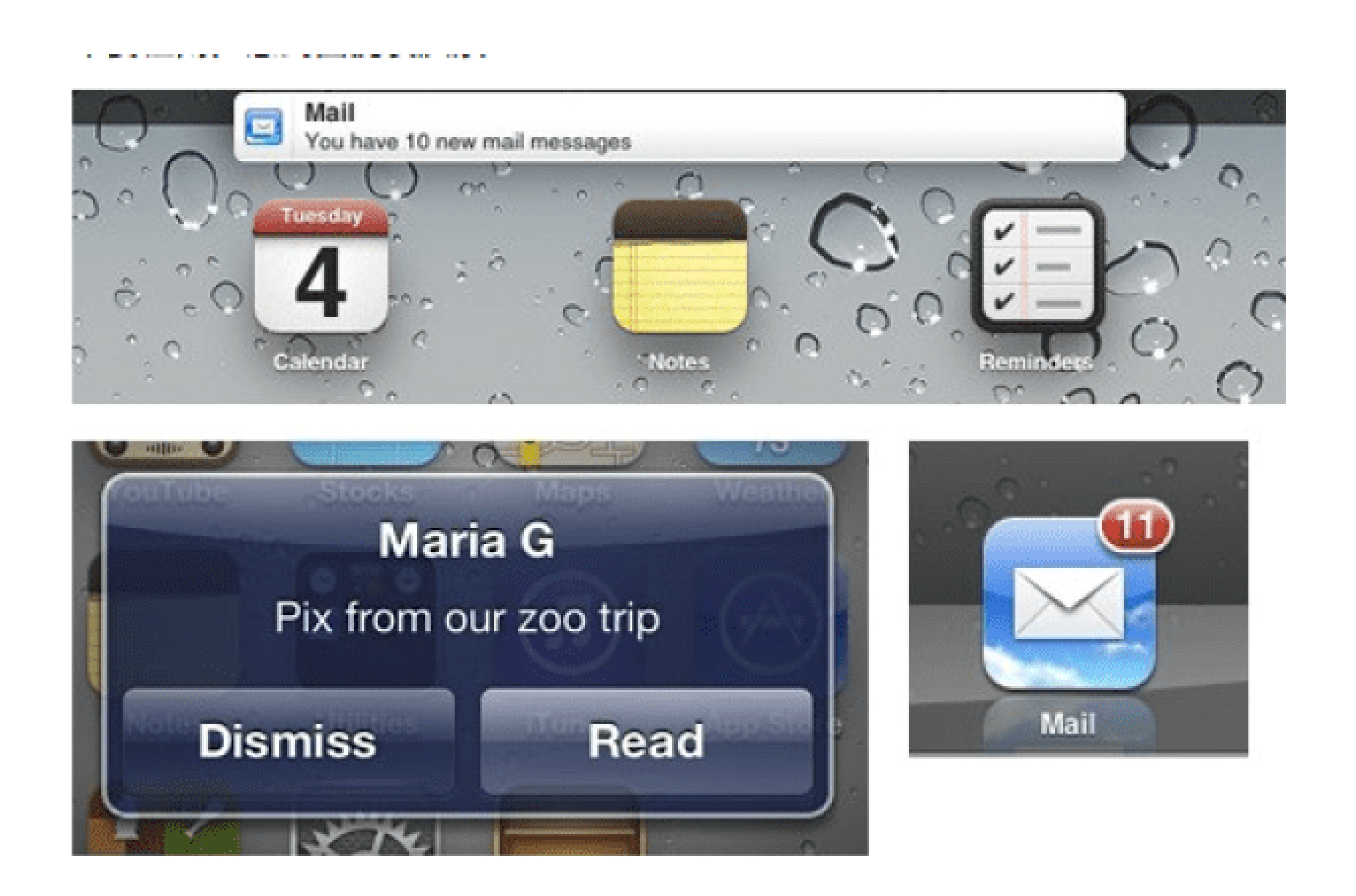
在移动互联网发展的初期，“推送”这个词很少出现在用户和从业人员眼中，随着移动互联网近乎疯狂的发展，推送变成了企业推广自己产品和保持用户粘性的不二选择，继而“推送”本身也被广大用户所知晓。

说到推送，我觉得大家应该都会有这样的体验——玩游戏的时候被推送所干扰。我自己很讨厌推送，尤其是很久不用的应用还在每天两三条地向我推送无节操的广告。言归正传，下面来看看 Android 系统和 iOS 系统上推送的展现形式。

Android 的推送：



iOS 的推送：





提示：当然自从 iOS7 的诞生，上下拉框的设计再也不只是 Android 的亮点了，世界也从此扁平化了起来。



提示：两个系统的推送原理有一定差别，但阐述推送的原理不在本书的范围内，有兴趣的朋友可以自己多看看相关资料。有个有趣的现象，我认识的很多测试工程师在测试的过程中最不关心两个测试点，一个是 iOS 的越狱系统，另一个就是推送。大部分测试工程师在测试过程中总会主动触发推送，从而满足测试的需求，其实仅仅这样做是不够的。

我们来看一下在测试推送的时候需要关注哪几点：

- 智能机在关机、待机、打开等状态下执行推送的功能、消息显示以及推送跳转等是否正确？
- 应用在打开、未打开状态，应用启动且在后台运行等情况下，查看推送的功能、消息显示以及推送跳转是否正确？
- Android 系统和 iOS 系统虽然都有推送这种功能，但是使用的机制完全不同，需要对两者同时关注多次推送以及推送的成功率。

- 推送本身分成主动推送和被动触发推送(某些业务场景触发的回调推送)。很多测试工程师在测试的时候，为了图方便，会让服务器直接发送推送从而测试这个功能，其实即使通过这样的测试，依然无法确认回调的推送就是正确的，这一点要注意。
- 推送的消息在阅读前后，其标示消息数量的数字是否改变?
- 单条或多条推送的文字显示以及跳转界面是否正确?
- 多语言系统环境下，推送的本地化翻译信息是否显示正确?

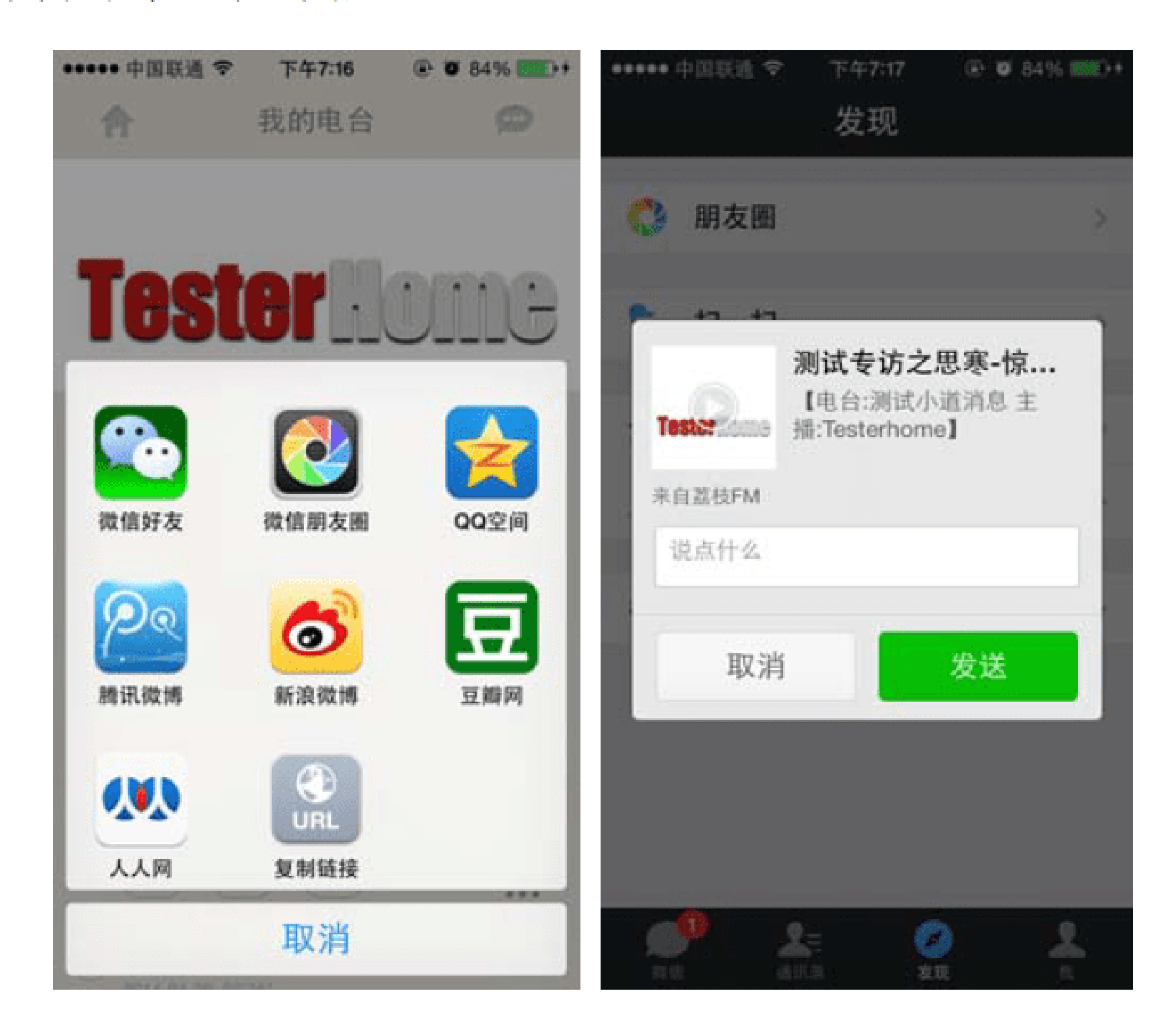
以上列出了我认为相对重要的几个测试点，实际上肯定不仅限于此。推送真的可以说是移动互联网发展历史上一个重要功能，现在无论什么类型的应用都会带有推送。推送不仅仅满足了用户的虚荣心（被别人关注的心理），甚至还能让一些应用死而复生，着实有四两拨千斤的功效。

4.10 分享跳转

关注我微博的人都能够看出我的习惯——上班下班时间不停地在微博和微信上分享Zaker和在“知乎日报”发布当前我觉得有价值的内容。如果感觉烦可以取消关注哈(我微博是 Monkey 陳瞱瞱)。

随着时代的发展，我们的社交圈越来越杂，各种小的创业公司要傍着这些“大腿”才能够推广自己的产品，于是“分享”这个功能就默默地产生了。我们先来数数有哪些“大腿”，最常见的是阿里系产品，企鹅系产品等，国外耳熟能详的就是 Twitter 和 FaceBook（脸谱）。自此以后我们就能够看到一个现象，原本一个人只要发一个状态或者图片到社交圈，现在则需要同时在自己所有的社交圈发布一遍。分享也不是银弹，有些不能同步的就得自己发送好几遍。不过曾经也有文章写过，人类的确对于“晒”这样一种行为始终乐此不疲。说实话，我老婆要是读到这段文字肯定会想:“你就是那种最喜欢晒的人，还在这里说别人”。

回归主题，分享这个功能由于各个应用的做法不同而有不同的表现形式。有的分享需要在应用之间跳转，比如微信：



有的直接分享即可成功，比如新浪微博：



在使用应用分享功能的时候，往往会有分享不成功的情况，并且大部分应用都不会很明确地告诉用户原因是什么，用户都是事后才发现自己没有分享成功，这可能是账号绑定 token 过期造成的。这使得在移动应用测试分享的过程中，模拟 token 过期就变得非常重要，同时还要关注以下几个要点：

- 同时关注 Android 和 iOS 两个平台的分享功能。
- 如果分享之后的文案有动态变化，需要加以关注。
- 分享到微博、微信等应用之后，关注在这些平台上消息的来源以及点击之后的跳转是否正确，如下图所示中的“支付宝钱包”。

今天 21:52 来自支付宝钱包 | 举报

4.11 小　结

由于移动互联网应用实在太多，无论是类型还是功能都不是个人经验所能够覆盖的，所以本章仅从我自己经历过的应用上进行总结，旨在给移动应用测试的新人一个引导方向。相信看完本章之后你对于移动应用的测试点基本会有一些了解，如有问题可以随时反馈给我，你的反馈将会是对我最大的支持和鼓励。

第 5 章　常用工具介绍和实践

移动互联网发展至今，无论是 Android 还是 iOS 官方的文档还是第三方开源的工具层出不穷。在日常的沟通中，经常听到有很多人在问："××工具怎么用？"。首先我觉得关于工具怎么用这类问题自己去谷歌搜答案即可，其次使用工具只是开始学习的第一步，更多的是需要我们去了解工具的实现原理，以帮助我们更好地使用工具来达到测试的目的。

本章会在面上介绍移动互联网测试时常用的工具，同时会加入作者自己的实践经验和见解，包括功能测试、压力测试、性能测试等。

由于本书不是一本傻瓜式入门参考书，所以环境搭建和一些基础知识并不会在其中详细说明，如你遇到问题的话可利用谷歌搜索答案。此外，由于测试工程师并非纯技术类的工种，所以本书也不作为纯技术类的书籍，在原理上进行深究。

5.1 Monkey

Monkey 是 Android SDK 提供的一个命令行工具，它可以简单、方便地运行在任何版本的 Android 模拟器和实体设备上。Monkey 会发送伪随机的用户事件流，适合对应用做压力测试。

5.1.1 第一个简单的 Monkey 测试命令

现在同我刚做移动互联网应用测试的时候已经不同了，提到 Monkey 测试，大家都会觉得非常简单。接下来看一下 Monkey 测试的基本流程是怎样的：

选择被测试的机器或模拟器—>输入制定过策略的命令—>回车即可运行

Monkey 测试最简单的命令：

```
adb shell monkey -v 10
```

该命令指定了 Monkey 测试执行操作的次数，当成功运行了 Monkey 测试后，我们可以看到下图所示的日志（log）信息。

```
:Monkey: seed=0 count=10
:IncludeCategory: android.intent.category.LAUNCHER
:IncludeCategory: android.intent.category.MONKEY
// Event percentages:
//   0: 15.0%
//   1: 10.0%
//   2: 2.0%
//   3: 15.0%
//   4: -0.0%
//   5: 25.0%
//   6: 15.0%
//   7: 2.0%
//   8: 2.0%
//   9: 1.0%
//   10: 13.0%
:Switch: #Intent;action=android.intent.action.MAIN;category=android.intent.cate
com.sec.android.app.popupcalculator/.Calculator;end
    // Allowing start of Intent { act=android.intent.action.MAIN cat=[android.i
p.popupcalculator/.Calculator } in package com.sec.android.app.popupcalculator
:Sending Flip keyboardOpen=false
:Sending Touch (ACTION_DOWN): 0:(73.0,687.0)
:Sending Touch (ACTION_UP): 0:(73.19875,687.4951)
Events injected: 10
:Sending rotation degree=0, persist=false
:Dropped: keys=6 pointers=2 trackballs=0 flips=0 rotations=0
## Network stats: elapsed time=97ms (0ms mobile, 97ms wifi, 0ms not connected)
// Monkey finished
```

如果不指定命令中的－s 参数，那么 Monkey 测试默认的种子（seed）值等于零。接下来会显示操作事件的默认百分比，这些百分比数字对应的操作可参考 Monkey 测试的源码…/monkey/src/com/android/commands/monkey/MonkeySourceRandom.java 文件中的操作定义。

```
public static final int FACTOR_TOUCH        = 0;
public static final int FACTOR_MOTION       = 1;
public static final int FACTOR_PINCHZOOM    = 2;
public static final int FACTOR_TRACKBALL    = 3;
public static final int FACTOR_NAV          = 4;
public static final int FACTOR_MAJORNAV     = 5;
public static final int FACTOR_SYSOPS       = 6;
public static final int FACTOR_APPSWITCH    = 7;
public static final int FACTOR_FLIP         = 8;
public static final int FACTOR_ANYTHING     = 9;
public static final int FACTORZ_COUNT       = 10;    // should be last+1
```

在 Monkey 测试日志中还描述了 Android Intent 的切换情况、点击操作的坐标以及最后测试是否完成。在上图的例子中，Monkey 测试完全被执行，执行完成后会显示：

```
//Monkey finished
```

5.1.2　Monkey 测试工具实例

Monkey 工具还提供了很多参数，让测试变得多样化。我这里给出一个曾经用在某海外项目中的 Monkey 测试的命令。

```
adb shell monkey -p com.xxx.xxx -p com.xxx.xxx --pct-touch 30 --pct-motion
 30 --pct-trackball 0 --pct-nav 0 --pct-majornav 20 --pct-appswitch 10 --p
ct-anyevent 10 -s 12867 -v --throttle 300 20000>MonkeyTest.txt
```

显而易见，这个 Monkey 测试的命令相比上一个要复杂得多。它主要是针对一些操作事件做了限制，从而减少了 Monkey 伪随机化的无效操作。体现在：

- 使用-p 参数来指定测试应用的包名（Package）。
- 使用--pct-xxx 参数进行操作的限制。比如操作不仅仅需要点击，还需要滑

动、长按，在智能手机硬件按钮、应用之间的切换等。

- 使用-s 参数来指定命令执行的 Seed 值。
- 使用--throttle 参数来控制 Monkey 每个操作之间的时间间隔。

这里需要注意的是 Monkey 工具指定应用包名的时候，只支持 Android Activity 属性的包名，如指定非 Activity 属性的话会出现如下提示：

```
**No activities found to run, Monkey aborted.
```

我们再来看一个实例。

```
Adb shell monkey -p com.xxx.xxx -p com.android.mms --pct-touch 30 --pct-mo
tion 20 --pct-trackball 0 --pct-appswitch -s 12867 --ignore-crashes --igno
re-timeouts -v --throttle 300 20000>MonkeyTest.txt
```

在执行 Monkey 工具的测试，会因为应用的崩溃（Crash）或没有响应（ANR）而意外中止，所以需要在命令中增加限制参数--ignore-crashes 和--ignore-timeouts，让 Monkey 在遇到崩溃或没有响应的时候，会在日志中记录相关信息并继续执行后继的测试。

5.1.3 Monkey 测试日志查看

Android 的应用产生 Crash 的原因很多，这里我新建了一个 Android 的项目，实现点击按钮会改变 TextView 文字的功能，MainActivity 类的代码如下所示：

```
package com.example.crashapp;


import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;
import android.view.View;
```

```
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.TextView;

public class MainActivity extends Activity {

        private Button bt1;
        private TextView tx;
        private TextView tx2;
        @Override
        protected void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                setContentView(R.layout.activity_main);
        bt1 =(Button)findViewById(R.id.button1);
        tx = (TextView)findViewById(R.id.textView1);
        bt1.setOnClickListener(new OnClickListener(){

                @Override
                public void onClick(View v) {
                        // TODO Auto-generated method stub
                        tx2.setText("xxx");
                }
        });

        }
}
```

不难看出，bt1、tx 和 tx2 都已经创建了对象，按钮点击事件中的 tx2 并没有和资源

文件 R 中有任何的关联。所以点击按钮 bt1 之后，就会造成崩溃（crash）。我尝试针对这个小应用执行 Monkey 测试，命令如下所示：

```
applematoMacBook-Pro:~apple$ adb shell monkey -p com.example.crashapp -v 100
```

结果显而易见，马上就出现了 Monkey 测试崩溃的日志，如下显示：

```
// CRASH: com.example.crashapp (pid 28004)
// Short Msg: java.lang.NullPointerException
// Long Msg: java.lang.NullPointerException
// Build Label: generic/vbox86p/vbox86p:4.1.1/JRO03S/eng.buildbot.20130507.170553:userdebug/test-keys
// Build Changelist: eng.buildbot.20130507.170553
// Build Time: 1367939191000
// java.lang.NullPointerException
//       at com.example.crashapp.MainActivity$1.onClick(MainActivity.java:27)
//       at android.view.View.performClick(View.java:4084)
//       at android.view.View.onKeyUp(View.java:7669)
//       at android.widget.TextView.onKeyUp(TextView.java:5328)
//       at android.view.KeyEvent.dispatch(KeyEvent.java:2633)
//       at android.view.View.dispatchKeyEvent(View.java:7086)
//       at android.view.ViewGroup.dispatchKeyEvent(ViewGroup.java:1358)
//       at android.view.ViewGroup.dispatchKeyEvent(ViewGroup.java:1358)
//       at android.view.ViewGroup.dispatchKeyEvent(ViewGroup.java:1358)
//       at android.view.ViewGroup.dispatchKeyEvent(ViewGroup.java:1358)
//       at com.android.internal.policy.impl.PhoneWindow$DecorView.superDispatchKeyEvent(PhoneWindow.java:1892)
//       at com.android.internal.policy.impl.PhoneWindow.superDispatchKey
```

```
Event(PhoneWindow.java:1369)
//        at android.app.Activity.dispatchKeyEvent(Activity.java:2356)
//        at com.android.internal.policy.impl.PhoneWindow$DecorView.dispat
chKeyEvent(PhoneWindow.java:1819)
//        at android.view.ViewRootImpl.deliverKeyEventPostIme(ViewRootImp
l.java:3575)
//        at android.view.ViewRootImpl.handleImeFinishedEvent(ViewRootImp
l.java:3545)
//        at android.view.ViewRootImpl$ViewRootHandler.handleMessage(ViewR
ootImpl.java:2795)
//        at android.os.Handler.dispatchMessage(Handler.java:99)
//        at android.os.Looper.loop(Looper.java:137)
//        at android.app.ActivityThread.main(ActivityThread.java:4745)
//        at java.lang.reflect.Method.invokeNative(Native Method)
//        at java.lang.reflect.Method.invoke(Method.java:511)
//        at com.android.internal.os.ZygoteInit$MethodAndArgsCaller.run(Zy
goteInit.java:786)
//        at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:553)
//        at dalvik.system.NativeStart.main(Native Method)
//
** Monkey aborted due to error.
```

我们能够看到这是一个空指针（NullPointerException）错误，日志中有多处显示：

```
// java.lang.NullPointerException
```

一般在这类错误信息下会有很多方法的错误信息，我们需要关心的是和我们应用相关的信息。由于我的应用的包名是 com.example.crashapp，所以寻找到了这行错误信息：

```
//at com.example.crashapp.MainActivity$1.onClick(MainActivity.java:27)
```

这行错误信息很明确地告诉我，是该应用的 MainActivity 类中的第 27 行代码出现了问题。我们回溯该应用工程的第 27 行代码：

```
tx2.setText("xxx");
```

这里正是没有实例化的对象 tx2 第一次被引用的地方。

一般情况下，Monkey 测试的日志会非常长，我们需要进行有效的过滤，才不至于产生填写重复（duplicate）的缺陷。我们会对关键字 CRASH 进行过滤筛，选出所有的日志中出现崩溃记录，再对出现问题的所在类和代码行数进行过滤，从而更进一步筛选出不重复的日志，然后再进行缺陷的汇报。

5.1.4 Monkey 测试注意点

在我们执行 Monkey 测试的过程中有一些需要注意的事项。

1. 准备好一切测试的前置条件

比如手机需要安装哪些应用，SD 卡中需要哪些数据，Android 系统需要开启哪些特殊设置或功能等。这些工作尽量在进行 Monkey 测试前做好充分准备。一方面提前准备可以将执行 Monkey 测试所在的环境模拟得更真实，另一方面如果想在 Monkey 测试执行的过程中设置点什么那是非常困难的事情。

2. 慎重使用 adb 服务

执行 Monkey 测试一般就这样两种场景：工作时间和休息时间。如果是在休息时间自然问题不大，如果是在工作时间，你想在模拟器或真机执行 Monkey 测试的同时做其他很重要的工作（这的确是一个很聪明的想法），切记慎重使用 adb 相关命令，尤其是同一个 sdk 下的 adb kill-server。由于 Monkey 是通过 adb shell 命令启动的，当由于某种原因在你使用 adb 命令时重启 adb 服务的话，那么 Monkey 测试的日志记录就会被中止，但正在机器上执行的测试行为不会停止。换句话说，你会看到测试在执行，但却

没有任何记录了。如果发生这种事情，那么将是非常悲惨的记忆。

3. 同时记录 Android 系统日志

这在 Android1.5 时代的时候几乎是没有人在乎的事情。但是现在回头再看，有太多的问题不是仅仅依靠 Monkey 测试的崩溃日志，就能够定位到问题所在了，有些时候必须借助 Android 的系统日志，才能更全面地去了解问题。所以在执行 Monkey 测试的同时，至少我认为 adb logcat 是一件必须要做的事情。

4. 一定需要记录 Seed 值

Seed 值是唯一能够重现（repro）Monkey 测试中出现的崩溃问题的方法。当一个 Monkey 测试出现的问题被修复的之后，往往很多测试不知道如何进行验证，这时候只能通过相同的参数，相同的 Seed 值进行重现并验证。

5. 不要使用单一的命令

很多测试工程师认为该工具仅仅是一个命令行工具，非常简单。我只能说认为很简单的都是不会使用的。我建议多写几条有不同测试偏重点的 Monkey 测试命令，在一个项目中同时使用这些命令进行测试，以便在每次测试时达到不同的测试效果，找到更多的缺陷。在有限时间内，使用简单高效的方法去发现尽可能多的缺陷，不正是我们测试工程师所追求的吗?

6. 必须重视 Crash

我从 Android 1.5 时代用这个工具至今，已经算是一个老用户了。虽然 Monkey 测试有部分缺陷我们无法准确地得知重现步骤，但是通过长期的经验判断，Monkey 测试所出现的 NullPointException，都是可以在用户使用时出现的，何时出现只不过是时间问题。所以，从本质上来说，Monkey 所有的 CRASH 都需要在发布（release）前修复（fix）掉。

5.1.5 Monkey 工具再探索

前面我已经围绕 Monkey 测试工具的使用和注意点说了一些自己的实践经验。接下来我从该工具的另外一个切入点继续往下讲吧。

Monkey 测试工具本身到底是怎么实现检测的呢？从其伪随机流的操作来看，我相信大部分测试工程师都会觉得它的原理不会很复杂。事实也的确是如此，只不过我还是鼓励大家不要去“觉得”，还是去了解才会对测试真正有所帮助。

在 Monkey 测试工具中，有一个参数是--pct-trackball，这里就举个这个参数所对应的源码逻辑进行介绍，在 development/cmds/monkey/src/com/android/commands/monkey/MonkeySourceRandom.java 文件中可找到其核心代码如下：

```
Display display = WindowManagerImpl.getDefault().getDefaultDisplay();
//获取设备的屏幕尺寸
int dX = random.nextInt(10) - 5;
int dY = random.nextInt(10) - 5;
mQ.addLast(new MonkeyTrackballEvent(MotionEvent.ACTION_MOVE).addPointer
(0, dX, dY).setIntermediateNote(i > 0));
//随机生成滚轮滚动的坐标
```

从代码中很容易找到 Monkey 测试提供的每个参数，以及是如何实现的，有什么弊端。如果觉得 Monkey 测试的伪随机流还不能满足测试的需求，希望能够按照特定步骤来执行测试的话，可用有以下两种方法：

1. 更改 Monkey 自身的源码。在 Linux 环境下，下载要测试版本对应的全部源代码，在终端中定位到源代码的根目录，输入 make monkey 即可。通过编译之后可在/out/…/monkey/中获取 Monkey.jar 这个包。将编译好的包通过 adb push 到要测试的 Android 系统中的/system/framework 下。随后，或通过 Monkey 提供的参数启动，或通过自己写的参数调用即可。

2．可编写脚本进行指定的操作。写一个脚本，将需要 Monkey 执行的操作放在一个队列中，从而根据自定义的步骤执行。具体的操作我就不在本书中进行说明了。大家可自行谷歌相关信息，或者查看源码中的核心类：android-x.x.x /development/cmds/monkey/src/com/android/commands/monkey/MonkeySourceScript.java。

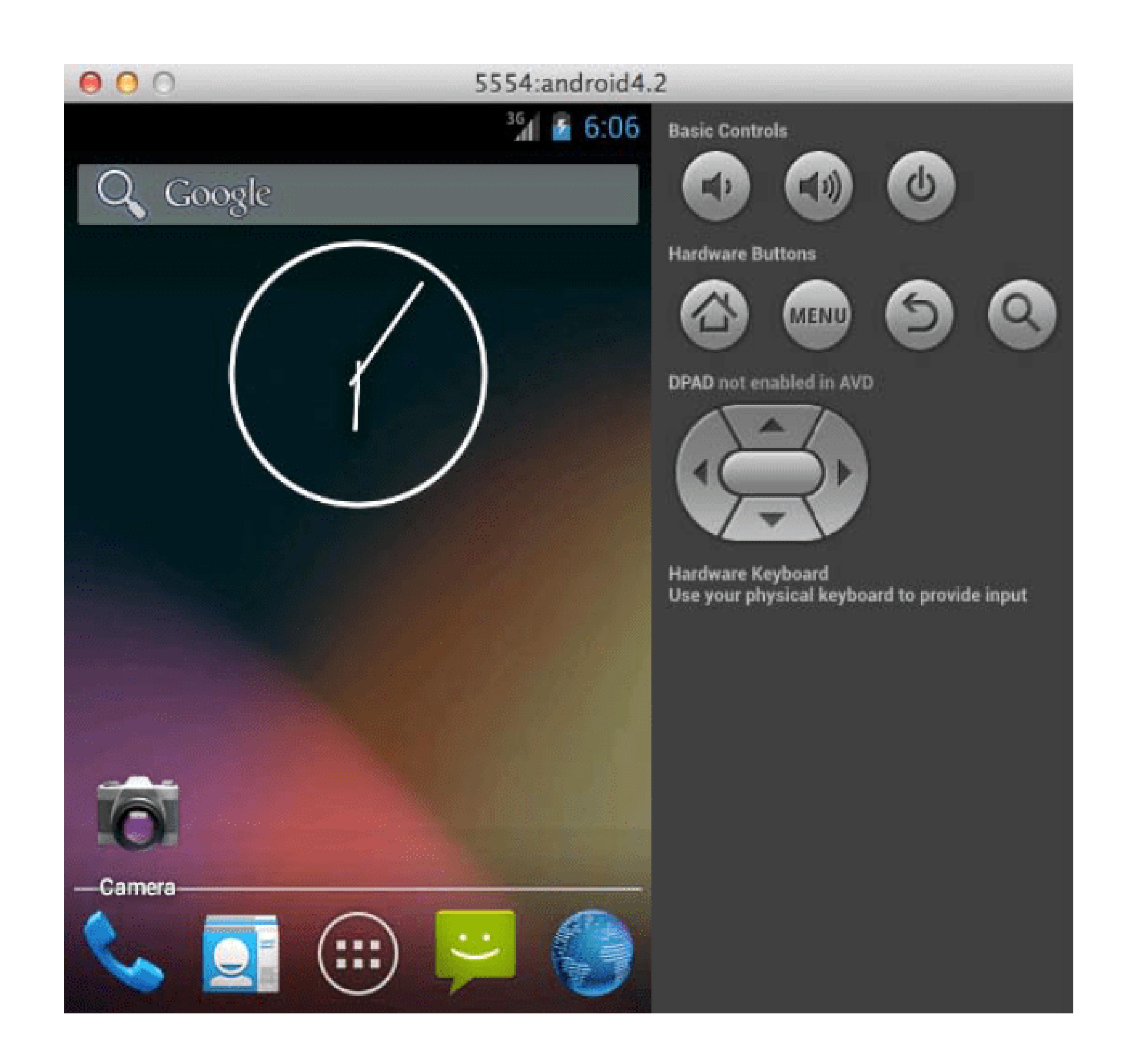
经过本小节的描述，相信你对于 Monkey 这个测试工具已经有了一个初步的、正确的认识。相信这远远没有满足你对应用测试的需求，更没有满足你的求知欲。其实这才是一个开始，你可以通过开源的文档继续学习。

5.2　Emulator

原本不想写模拟器的，但是最后考虑了一下还是写出来，不过这里并不是告诉大家模拟器是怎么启动的。

5.2.1　模拟器和真机的差异

对于 Android 模拟器，一般大家都会在 eclipse 或在终端中输入 emulator –avd <emulator name>来启动，如下图所示。



问题来了：模拟器和真机的差别到底有多少？模拟器是否可以代替真机呢？

接下来就具体的问题简单回答一下。

问题一：模拟器和真机差异有多少？

回答：模拟器是 Android 官方提供的一个可以模拟各种分辨率以及 Android 系统不同版本的虚拟机。我从 Android 1.5 时代就开始进行 Android 系统应用的测试，测试的项目都是基于客户定制机上的定制应用，所以在这方面有比较特别的经验。真机中也分成两种：Android 原生系统（比如 Nexus 系列）和厂商定制系统（比如锤子，索尼，小米等）。无论是哪种，首先可以肯定的是，模拟器无法进行 100%的模拟还原。模拟器本身在对某些功能的模拟上就存在先天的缺陷（比如相机、wifi 等一些系统界别的功能）。其次，在相同分辨率和系统版本下，模拟器和真机往往会在界面显示（UI）和某些按钮的响应上产生比较大的差异。总而言之，所有在模拟器上进行的测试都仅作参考，不能把对模拟器的测试和真机的测试划等号。

我们再来说一下 iOS 的模拟器。iOS 的模拟器和真机在界面显示上不会有差异（至

少我没有碰见过），不做任何越狱才能够支持的功能都无法在模拟器上运行。

问题二：模拟器是否可以代替真机呢？

事实上从问题一的回答已经得知，模拟器肯定是无法替代真机了。可是很多创业公司不可能拥有所有机型的真机，那又应该如何做测试呢？

回答：个人觉得这个问题基本上是每个测试人员都会碰见的问题，而 Android 测试和 iOS 测试由于系统和平台不同，还有着不同的困难。退一万步来说，哪怕公司有所有机型的真机，也没有这么多人手能够在每次项目迭代中测试完毕。

对此我有以下几条建议：

- 统计自己的应用被使用的数据

可以通过友盟或 Flurry 等在应用中的嵌入，得到应用在哪些机型上被安装了。从我自己的经验来看，排名前 10 位的机型基本上都是测试的重点，而且也都是市场上的主流机型。通过这么做可以过滤掉很大一部分安装量很小的机型。

- 可参考兼容性测试平台的测试结果

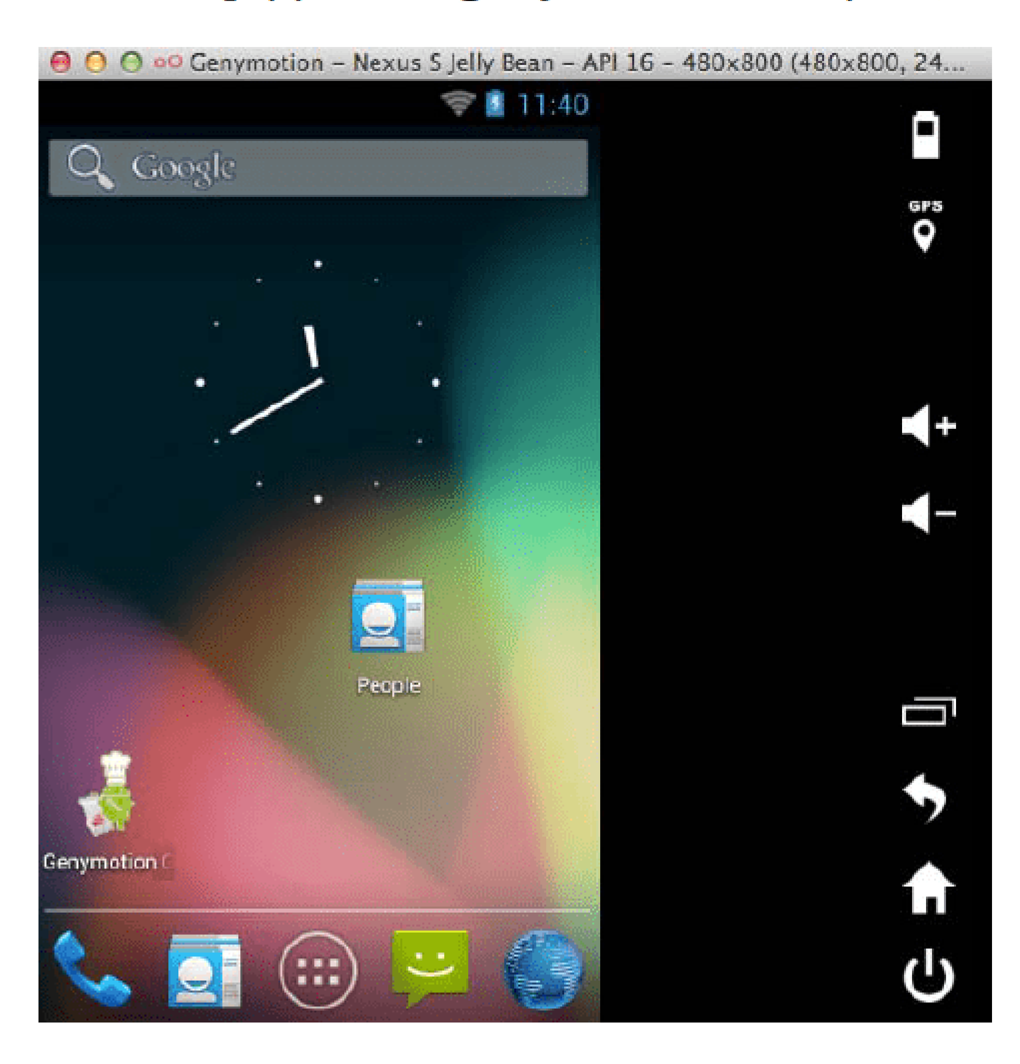
比如 Testin 或百度的 MTC 平台。虽然应用的兼容性测试不能完全依赖第三方开发的平台进行测试，但是可以起到一个参考的作用。从实际的效果来看，两者对于应用的安装、启动、卸载等兼容性测试做得还是很不错的。

- 保持和主流机型的用户的联系

这点很好理解。在正式发布前一个星期左右，可联系这些使用主流机型的用户，帮助进行一些安装、使用核心功能的测试。从实践经验来讲，这个方法我觉得是效果最好的。虽然大多数用户是小白，但是在使用内测版的时候还是非常积极的。对于 Android 的应用，直接发给这些用户使用即可，iOS 的应用可以通过 testflight 或申请企业级证书的方法，在正式上 app store 之前，让更多的测试工程师和用户可以进行测试。

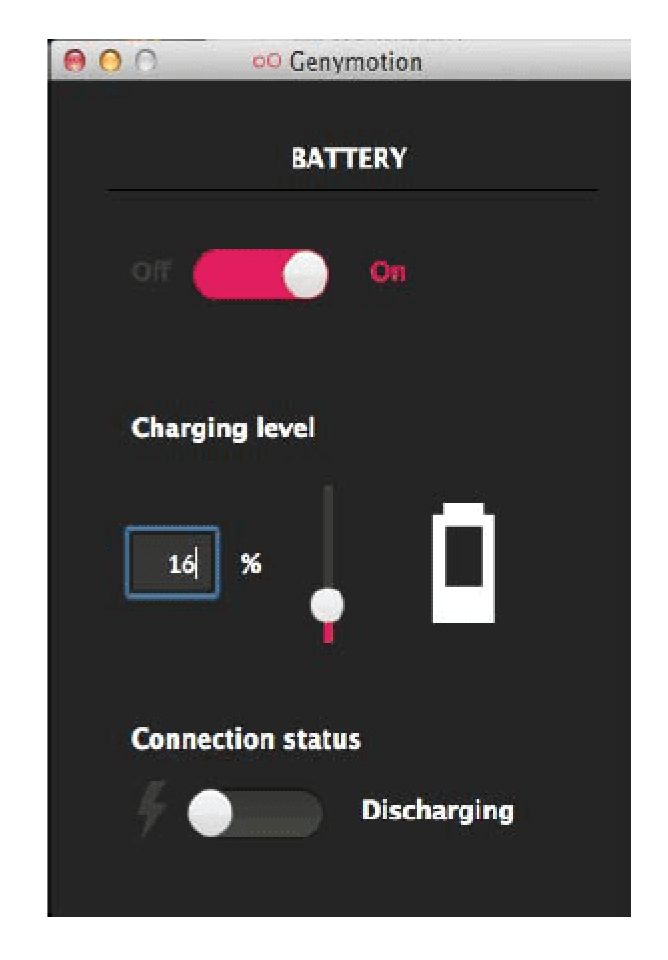
5.2.2 Genymotion

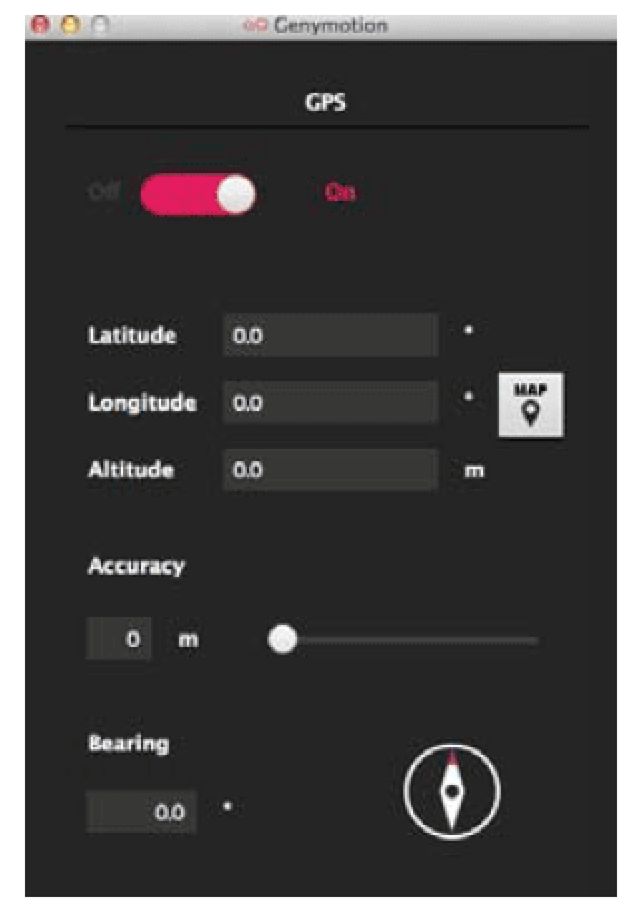
就在前不久，第三方发布了一个拥有极速性能的 Android 模拟器——Genymotion，如下图所示。其所有内容可在 http://www.genymotion.com/进行查看。



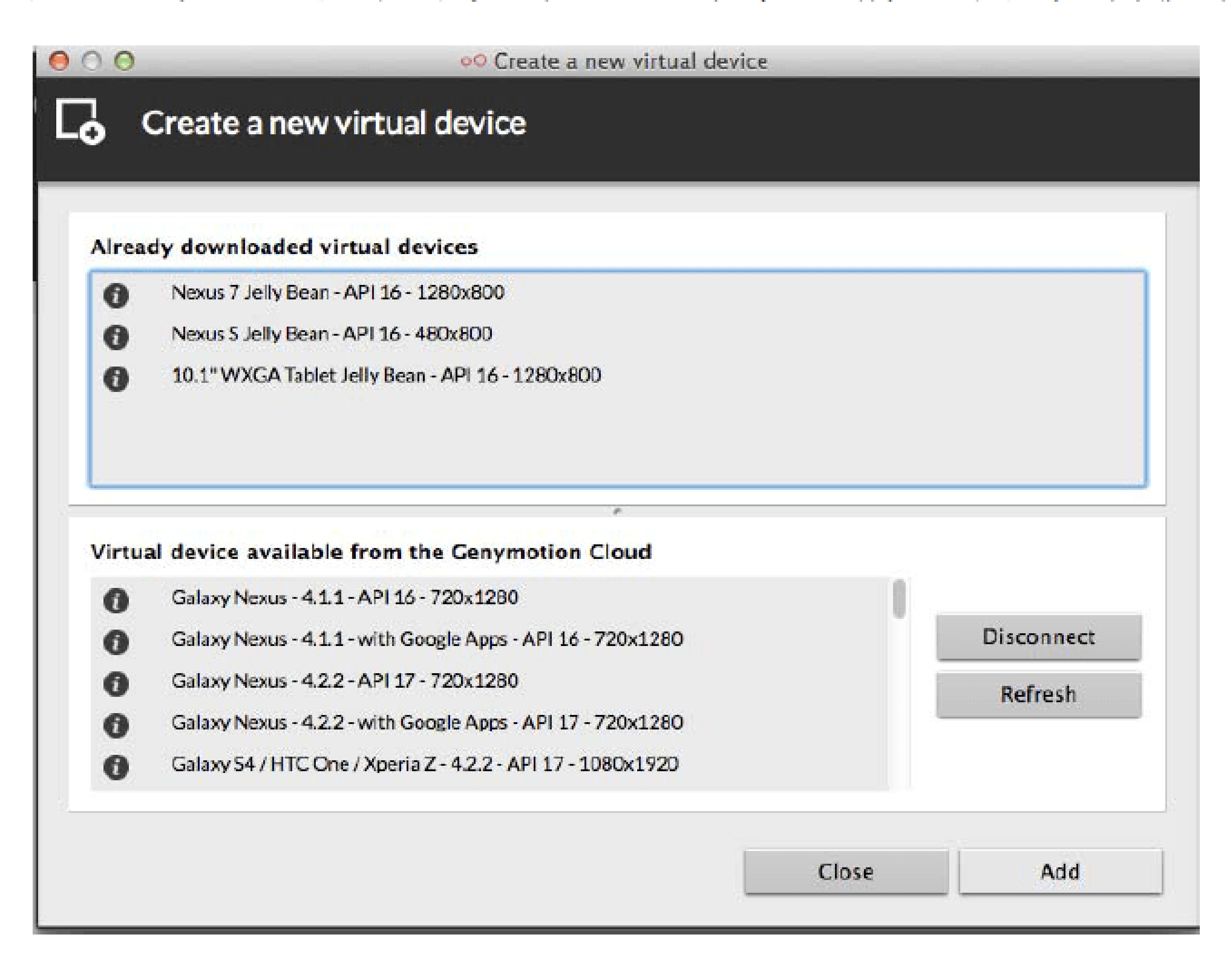
经过笔者亲自尝试之后发现，Genymotion 很好地支持了 Android 系统的基本功能，而且还是飞一般的性能，这一点令我非常惊讶。

同时 Genymotion 还支持界面化调节电量、以及调节 GPS 定位功能。如下图所示。





不过由于 Genymotion 目前还是 1.0.0 版，还是有一些缺陷的，一方面它只能模拟有限的手机型号，另外一方面（也是我比较重视的），目前还不能支持模拟器横竖屏。



5.2.3　模拟器常用功能举例

我们日常工作中，除了用模拟器模拟各种不同的 Android 版本系统以外，还会有一些特殊的交互，需要模拟器的支持。我在这里举几个最常见的例子。（由于模拟器支持的功能非常多，详尽内容可参见 Android SDK 文档中有关 emulator 的描述。）

我们来看一下模拟来电和收到短信。

打开 Android 原生的模拟器之后，输入以下命令来查看模拟器所在的端口：

```
adb devices
```

可以看到以下反馈：

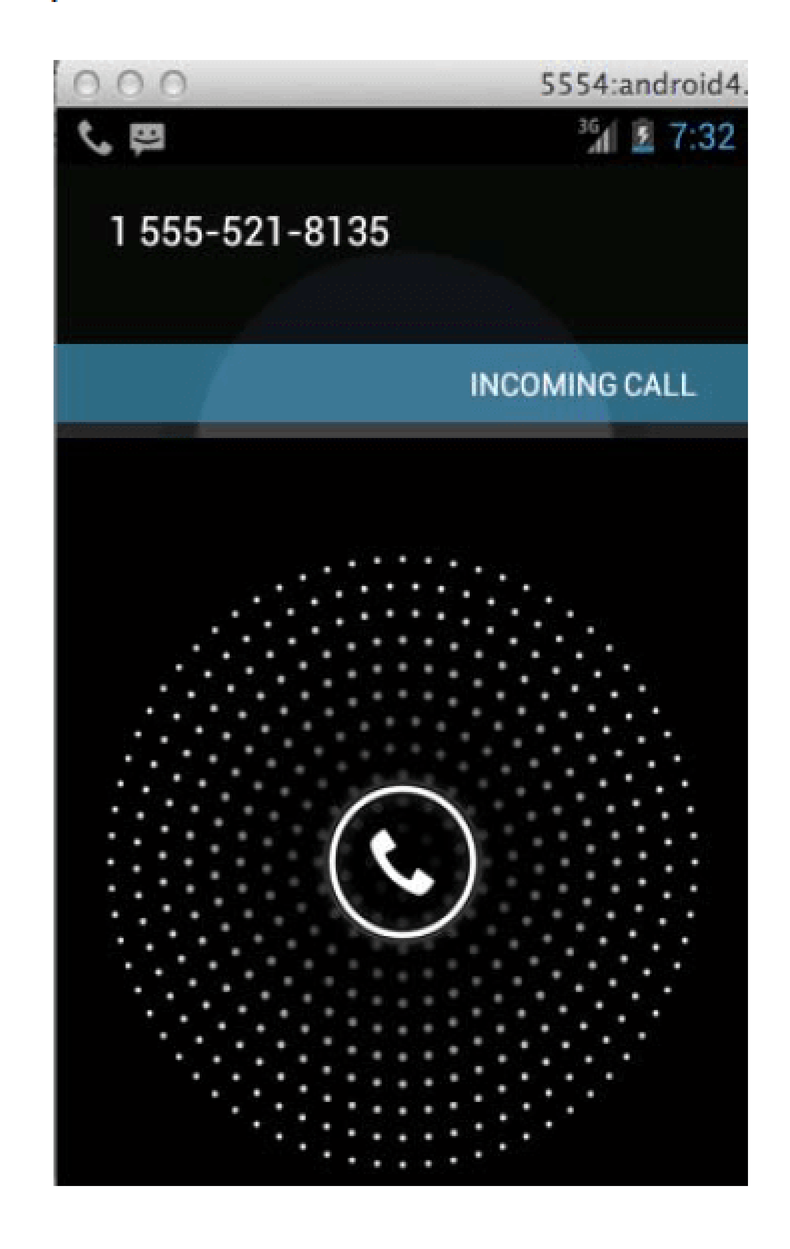
```
emulator-5554    device
```

接着通过输入 telnet localhost 5554 来和模拟器连接，连接成功的话，会显示如下

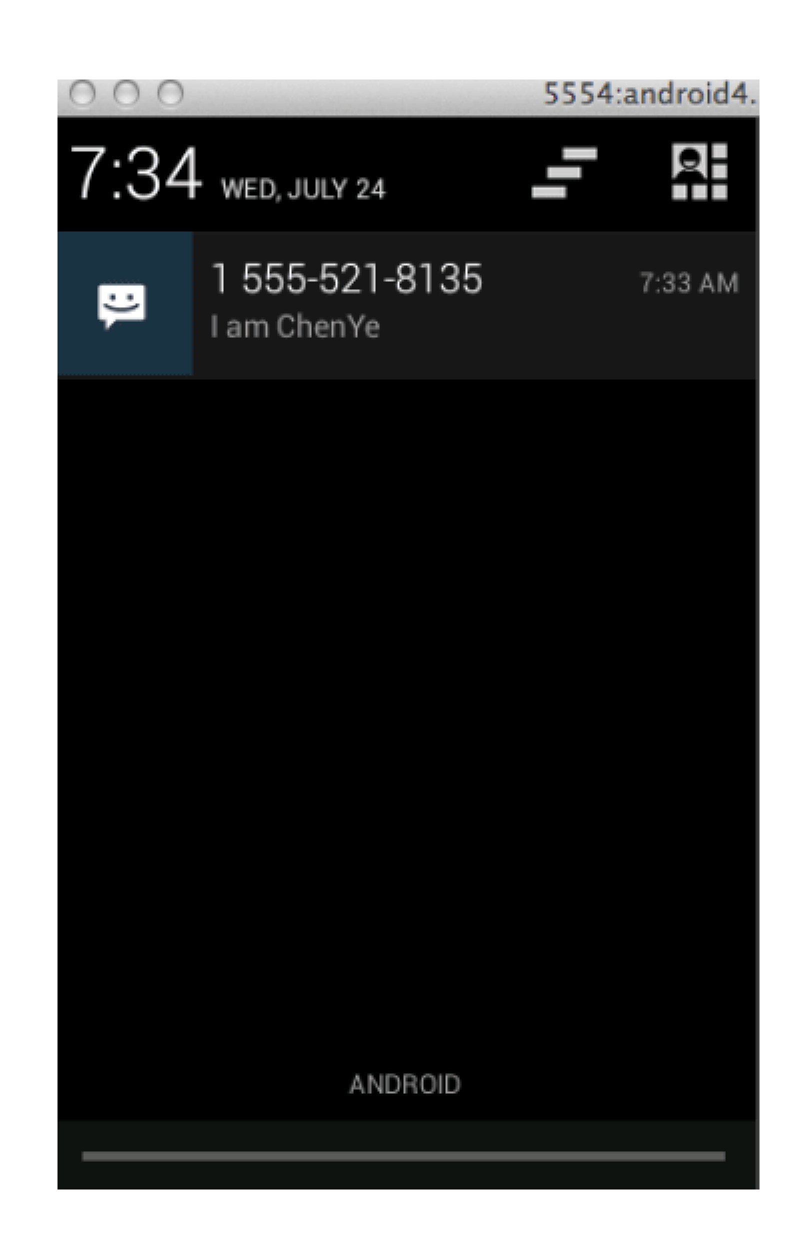
日志信息：

```
telnet localhost 5554
Trying ::1...
telnet: connect to address ::1: Connection refused
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
Android Console: type 'help' for a list of commands
```

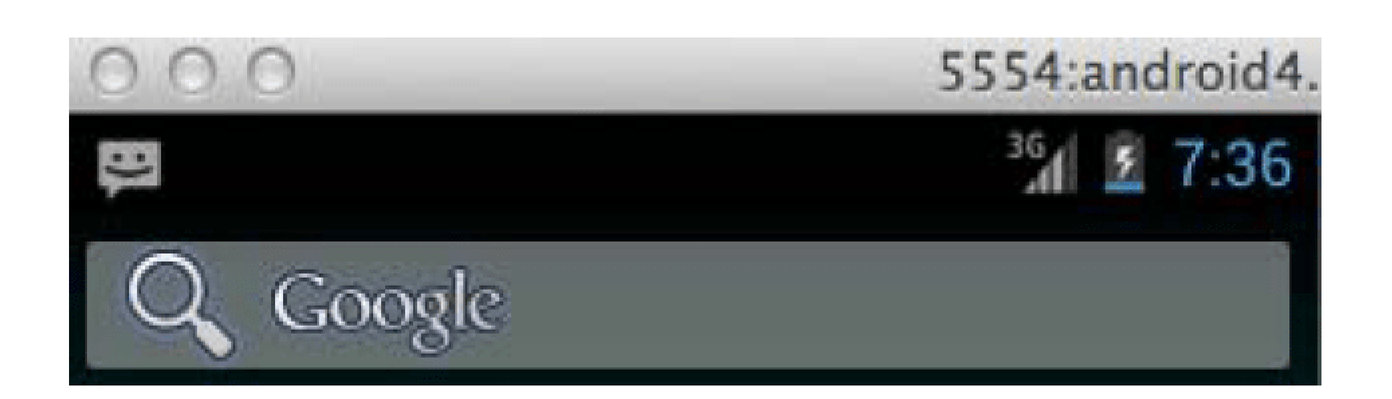
相关的命令都可以通过输入 help 来查询，非常方便。如果我们需要打一个电话给模拟器，那么需要输入 gsm call 15555218135（模拟器默认的号码是 15555218135），马上会看到模拟器有如下显示。



同样地，通过输入：sms send 15555218135 I am ChenYe，模拟器马上收到发来的短信，如下图所示。



在模拟器上模拟机器电量也是非常常见的场景。在 telnet 之后的模式下输入 power capacity 10，能够将模拟器的电量变为仅剩 10%的状态。



5.3　MonkeyRunner

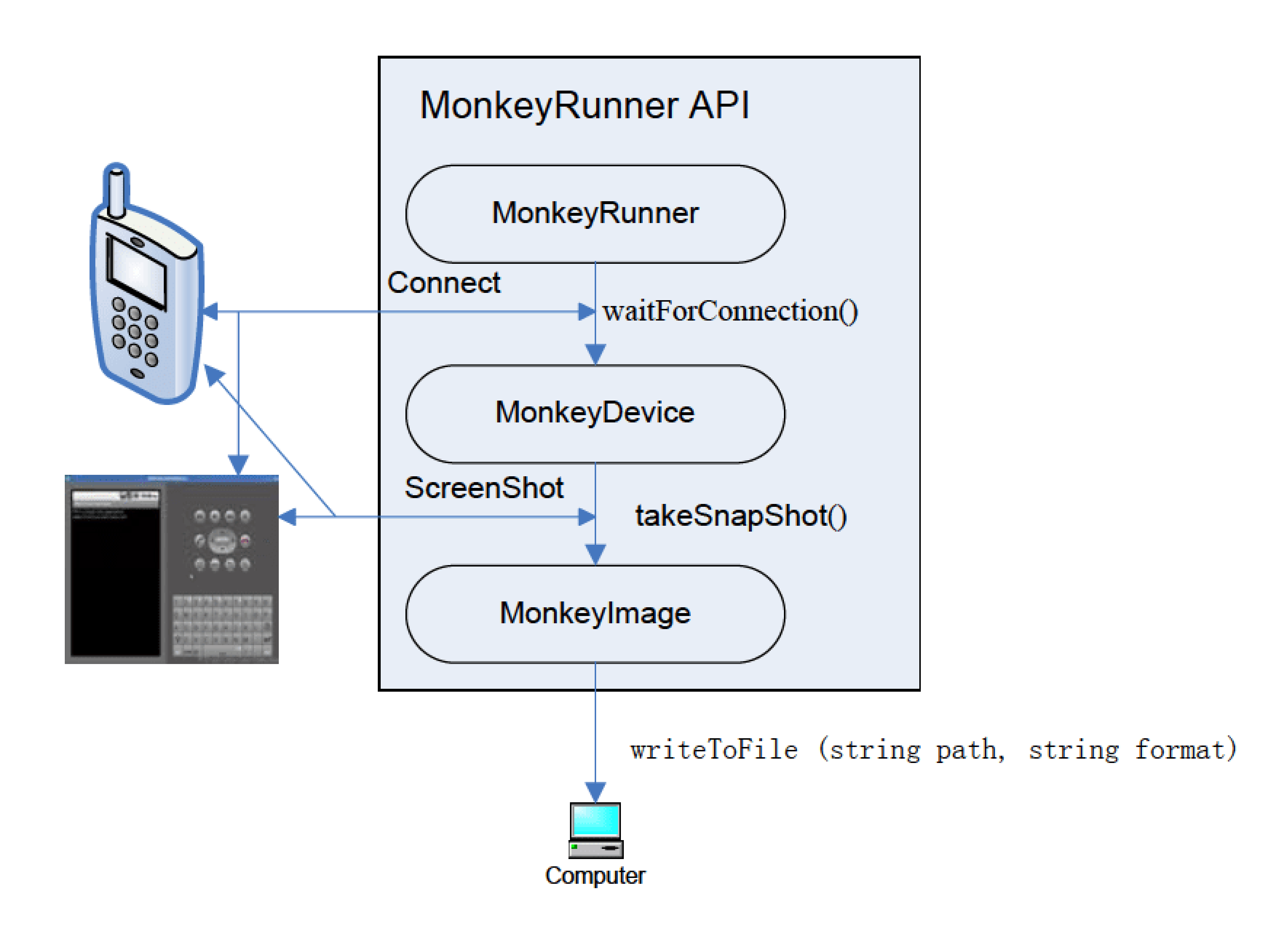
MonkeyRunner 是 Android SDK 原生提供的一个能够从外部控制 Android 设备或模拟器的工具。在 MonkeyRunner 中你可以向 Android 系统发送一些基本的事件，还可以进行截图。

MonkeyRunner 提供了 3 个可使用的包：MonkeyDevice、MonkeyImage 和 MonkeyRunner。我们可以引入这 3 个包之后，编写 python 脚本来执行 MonkeyRunner

测试。

MonkeyRunner 在 Android SDK 中有一个非常详细的例子，有兴趣的朋友可以去对照着例子自己实践一下，具体内容参照：http://developer.android.com/tools/help/monkeyrunner_concepts.html。

我自己一直在做 Android 第三方应用的测试，所以使用 MonkeyRunner 的时间比较少，用于应用界面的对比的回归测试和一些压力测试。基本流程如图所示。



看过 MonkeyRunner 原有的 API 的朋友估计都会感觉功能很少，事实上 MonkeyRunner 本身是支持使用第三方封装的方法库。在大部分的企业中，MonkeyRunner 更多地被用在对 Android 整机的测试中。在这类测试中，由于应用都是系统内置的并且测试用例涉及的应用众多，使用其他测试工具或框架都比较难实现，所以最终都会选择二次开发 MonkeyRunner 来满足公司对测试的需求。

在 MonkeyRunner 的 MonkeyImage 库中，有这样一种方法，它对验证测试结果起到了至关重要的作用，即 sameAs（MonkeyImage other，floast percent）。

boolean	sameAs (MonkeyImage other, *float* percent) Compares this MonkeyImage object to another and returns the result of the comparison. The percent argument specifies the percentage difference that is allowed for the two images to be "equal".

从其文档中对该方法的描述来看，它能够对比两张图片显示的结果是否一致。所以，一般使用这个方法来验证所有的测试执行完毕之后，实际结果和我们期望的结果是否一致。这里我们还看到一个参数 percent，这个参数可能会给我们造成一定的困惑。难道对比两张图片是否一致不应该设置成 100%吗?

从我之前的实践经验来看，如设置成 100%的话，最终结果是几乎都通不过。原因是，在真实的测试过程中，无论是真机还是模拟器，都会有一些不可控因素，会影响到图片某些像素的显示，比如电量状态、时间、通信信号等，所以才在这个方法中给出了百分比这个参数。但是如果不设置成 100%的话，又对比较出来的结果不那么放心。于是在应用的测试过程中，我在应用中增加以下代码，从而绕过了这个问题。

```
getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN, WindowMa
nager.LayoutParams. FLAG_FULLSCREEN);
```

这段代码的作用是让应用启动之后变成全屏（也就是不显示系统的状态栏），这样一来，设置成 100%图片对比就变得有可行性了。

在 Android 的项目中，让很多测试工程师头疼的是 Android 应用存在很多渠道包(对渠道包不了解的话就自行谷歌吧)。从形式上说，渠道包其实就是修改 mainfest.xml 中的一行字符串。

```
<meta-data android:name="渠道标签" android:value="渠道名" />
```

在其他功能上没有任何改变。但是对于测试工程师而言，所有需要发布的应用包都是需要经过回归测试的。对渠道包的回归可以借助 MonkeyRunner 这个测试工具来做，这里给出部分测试代码。至于 MonkeyRunner 的截图、对比等功能，有兴趣的朋友自己加上即可。python 代码如下：

```
import os
```

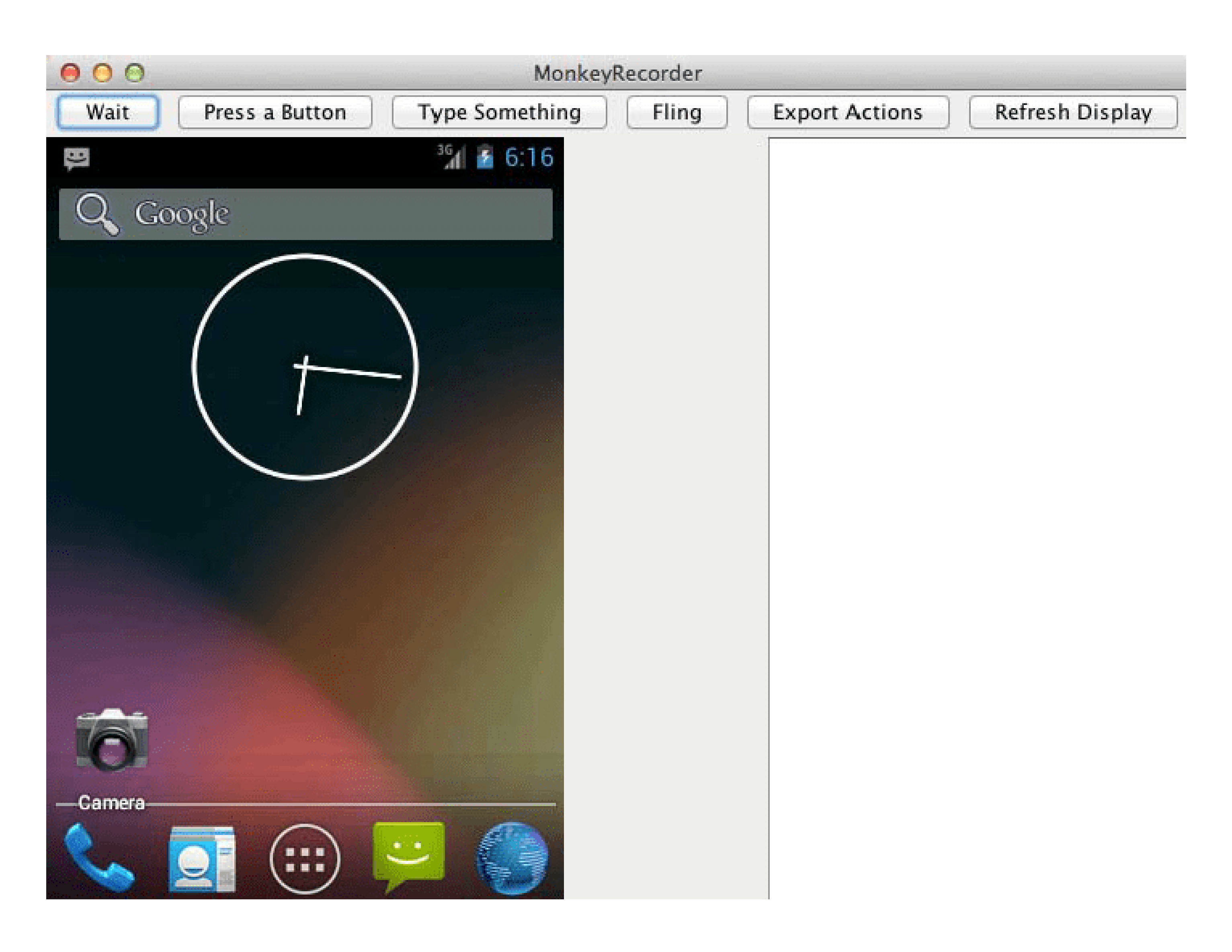
```
//遍历文件夹文件
def traverse(self,path):
        Application_apks = []
        for root, dirs, files in os.walk(path):
                for fn in files:
                        Application_apks(os.path.join(root,fn))
return Application_apks
//指定渠道包所在路径
Application_apkLists =  traverse('/Users/monkey/xxxx')
//执行安装，启动，卸载操作
for i in range(len(Application_apkLists)):
        try:
                os.system('./adb install %s')% Application_apkLists[i]
                os.system('am start -n com.xxx.xxx/com.xxx.xxx.xxx')
        except e:
                os.system('./adb unkaihuiinstall com.xxx.xxx')
                print 'The error in here'
```

MonkeyRunner 测试工具还支持录制回放功能。编写以下代码并保存成 mr_rocorder.py 文件。

```
from com.android.monkeyrunner import MonkeyRunner as mr
from com.android.monkeyrunner.recorder import MonkeyRecorder as recorder

device = mr.waitForConnection()
recorder.start(device)
```

连接好真机或模拟器之后，在终端输入 monkeyrunner mr_recorder.py，可看到如下显示的界面：



然后点击左边的屏幕和上面的操作事件按钮，即可执行对操作的录制，录制完毕可将脚本直接导出，产生类似如下的代码：

```
TOUCH|{'x':359,'y':17,'type':'downAndUp',}
TOUCH|{'x':156,'y':448,'type':'downAndUp',}
TOUCH|{'x':213,'y':448,'type':'downAndUp',}
TOUCH|{'x':210,'y':341,'type':'downAndUp',}
```

同样地，MonkeyRunner 既然支持录制，就肯定会支持回放，关于 monkeyplayback.py 文件的代码，有兴趣的朋友可以自行谷歌。由于网上都有现成的代码，故不在这里复制粘贴了。

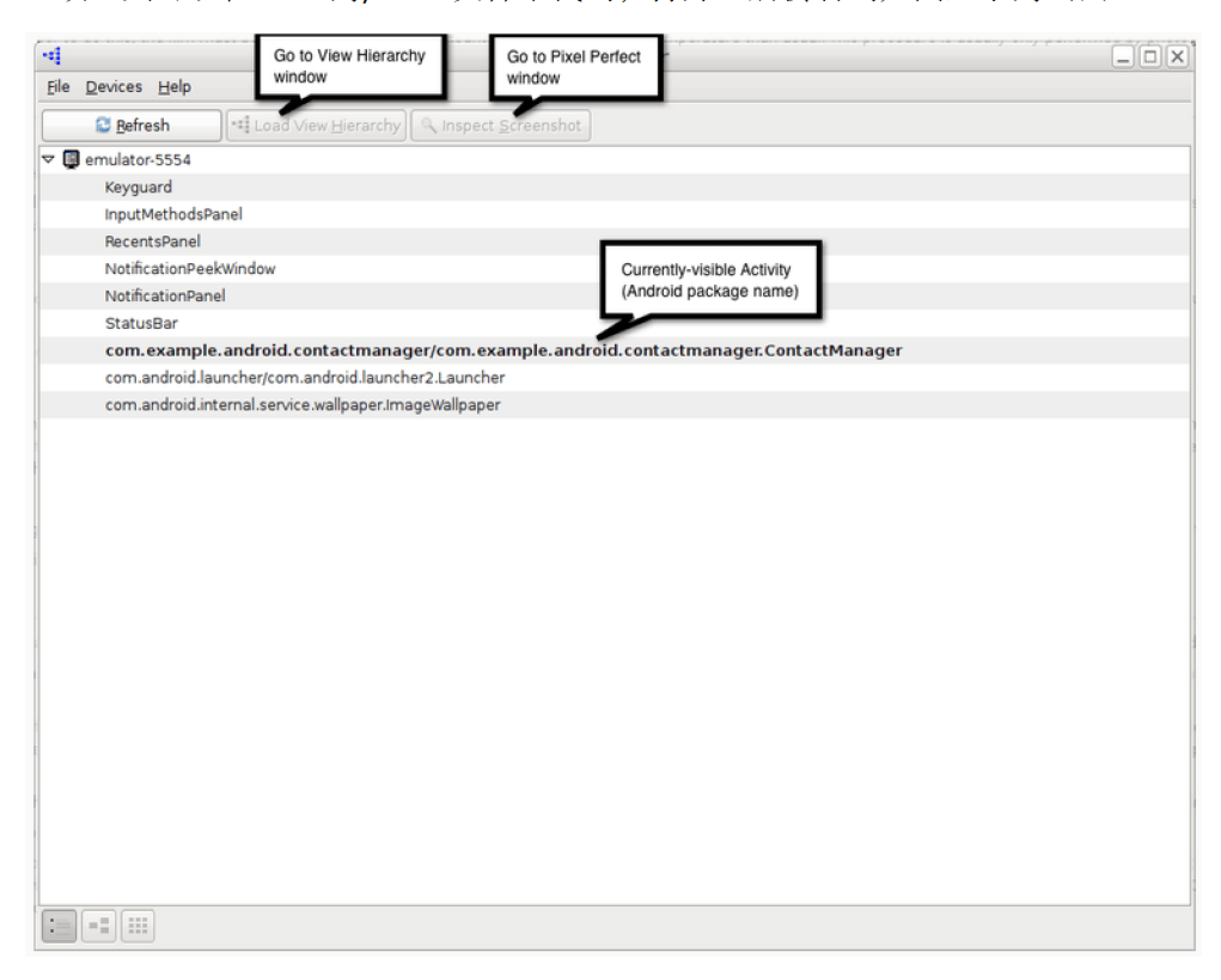
之前已经提到过 MonkeyRunner 其实更适合做系统级别的回归测试、压力测试。就我沟通的结果，高通、索爱等做整机测试的时候，使用该工具较多，更多的功能需自己扩展 API 来满足测试需求。

5.4 Hierarchy Viewer

做过 Android 相关工作的朋友应该对这个工具不陌生。其实在不解读源码的情况下，我是不想在本书中提及该工具的，因为都是谷歌里搜得到的知识。但是，毕竟本章写的是常用工具，所以在这里简要地单描述一下它的功能，熟悉该工具的朋友可以略过。

Hierarchy Viewer 是 Android SDK 原生提供的查看和优化应用界面的工具。它支持在没有应用源码的情况下，查看应用的视图结构以及放大每个界面，以便审视上面的每个像素。

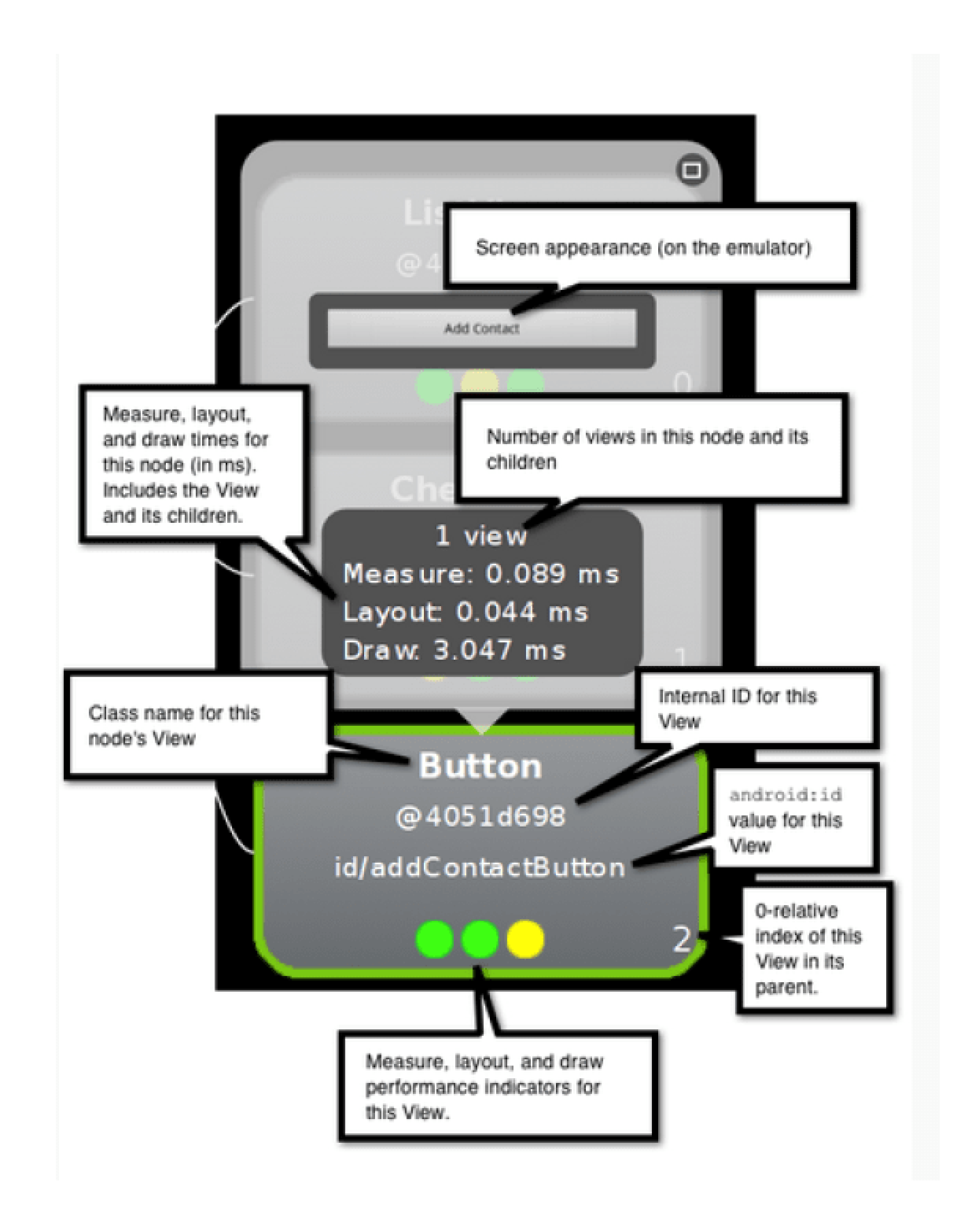
该工具可以在 SDK 的/tools 文件下找到，打开之后会看到如图显示的画面。



选中当前界面的 Activity 之后，点击上方的“Load View Hierarchy”按钮，可看到类似下图的画面。



从上图可看到，左边是该应用界面的结构树状图，右边从上到下分别是树状结构的缩略图、属性框以及树状结构图中元素在手机界面上的显示结果。在树状结构中，选中一个元素之后，会有如下图提示出现。



从图中我们能够看到有非常多的信息显示出来，最常用的有这样 3 种：

- 该控件的类型。比如图示是一个 Android Button 控件
- 该控件的 Id。在很多做界面自动化的时候经常会用到 Id，比如 Android 中的 FindViewById()方法等。图示 id 是 addContactButton。
- 该控件在 Android 绘制 View 的 3 个过程的消耗，分别是计算大小（Measure）、布局计算（Layout）和屏幕绘制（Draw），如图中所示。Hierarchyviewer 不仅仅能够将 3 个过程中的耗时显示出来，还会直观地在树状结构图中用红色、黄色、绿色表示绘制 View 的消耗。

Hierarchyviewer 主界面上还有一个 Inspect Screenshot 按钮，单击该按钮之后可放大画面，方便查看当前 Activity 界面的每个像素的详细信息。



提示：更多关于 Hierarchyviewer 的详细使用方法，在 Android SDK 的文档中都有描述，可参考 http://developer.android.com/tools/debugging/debugging-ui.html。这里就不多做说明了。

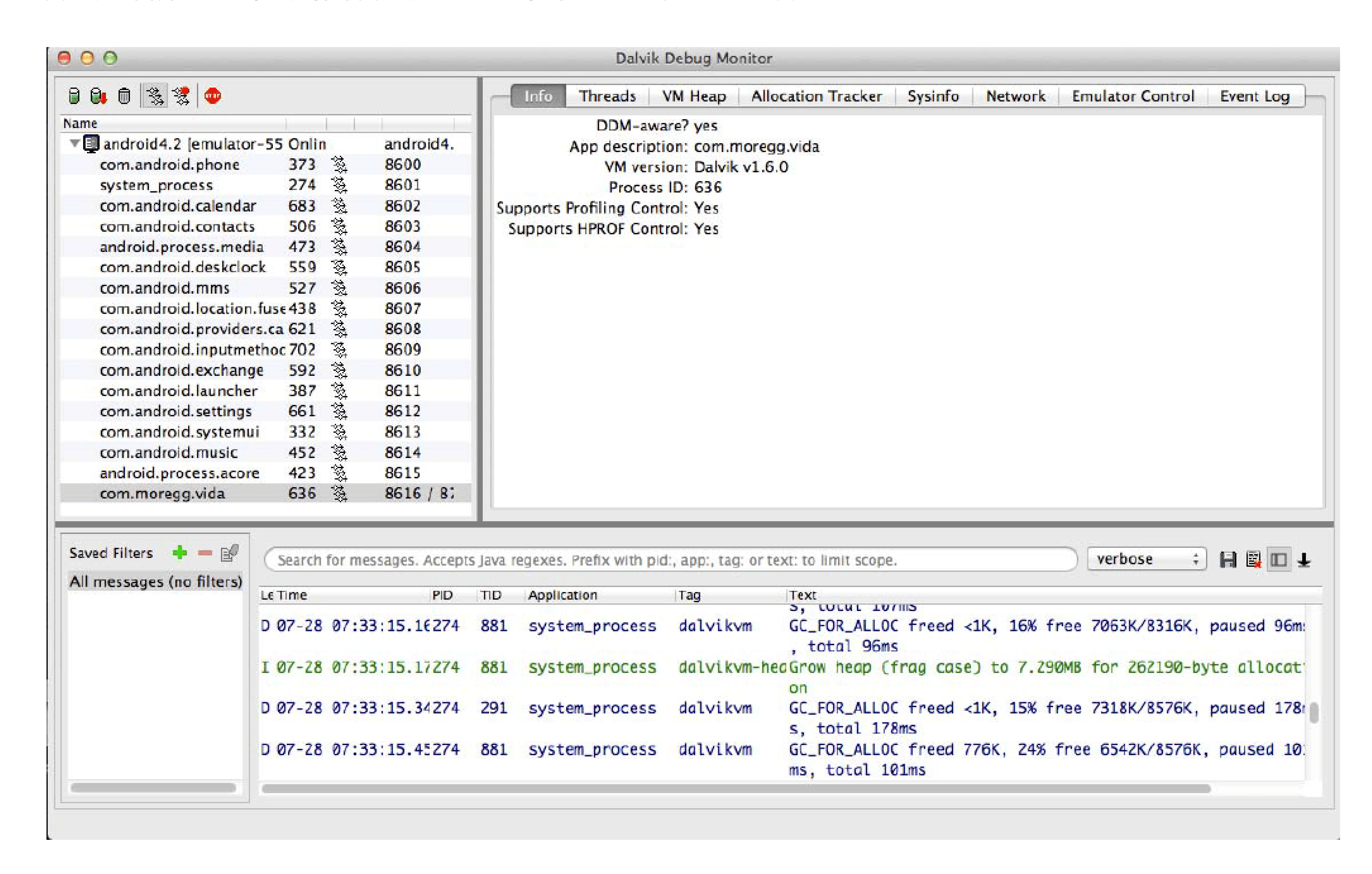
5.5　DDMS

DDMS 的全称是 Dalvik Debug Monitor Service，它可以提供诸如：为测试设备截屏、针对特定的进程查看正在运行的线程以及堆信息、Logcat、广播状态信息、模拟电话呼叫、接收 SMS、虚拟地理坐标等。



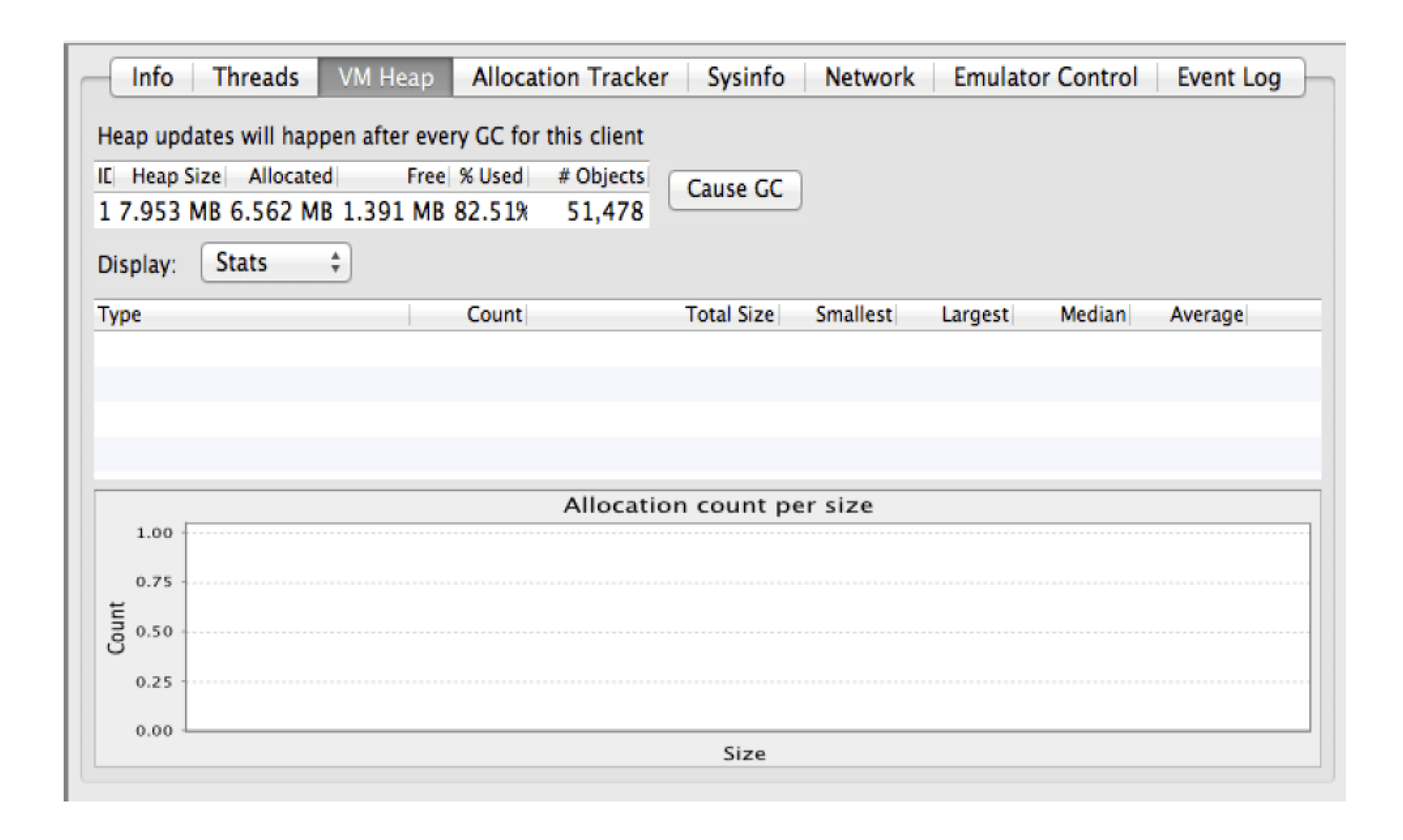
提示：在这里为大家举两个在测试活动中经常用到的功能。如想了解更详细的功能使用可参考 http://developer.android.com/tools/debugging/ddms.html。

DDMS 同样可以在 Android SDK 的/tools 下找到其启动程序。连接真机或在模拟器打开的情况下，我们打开 DDMS，其界面如下图所示。

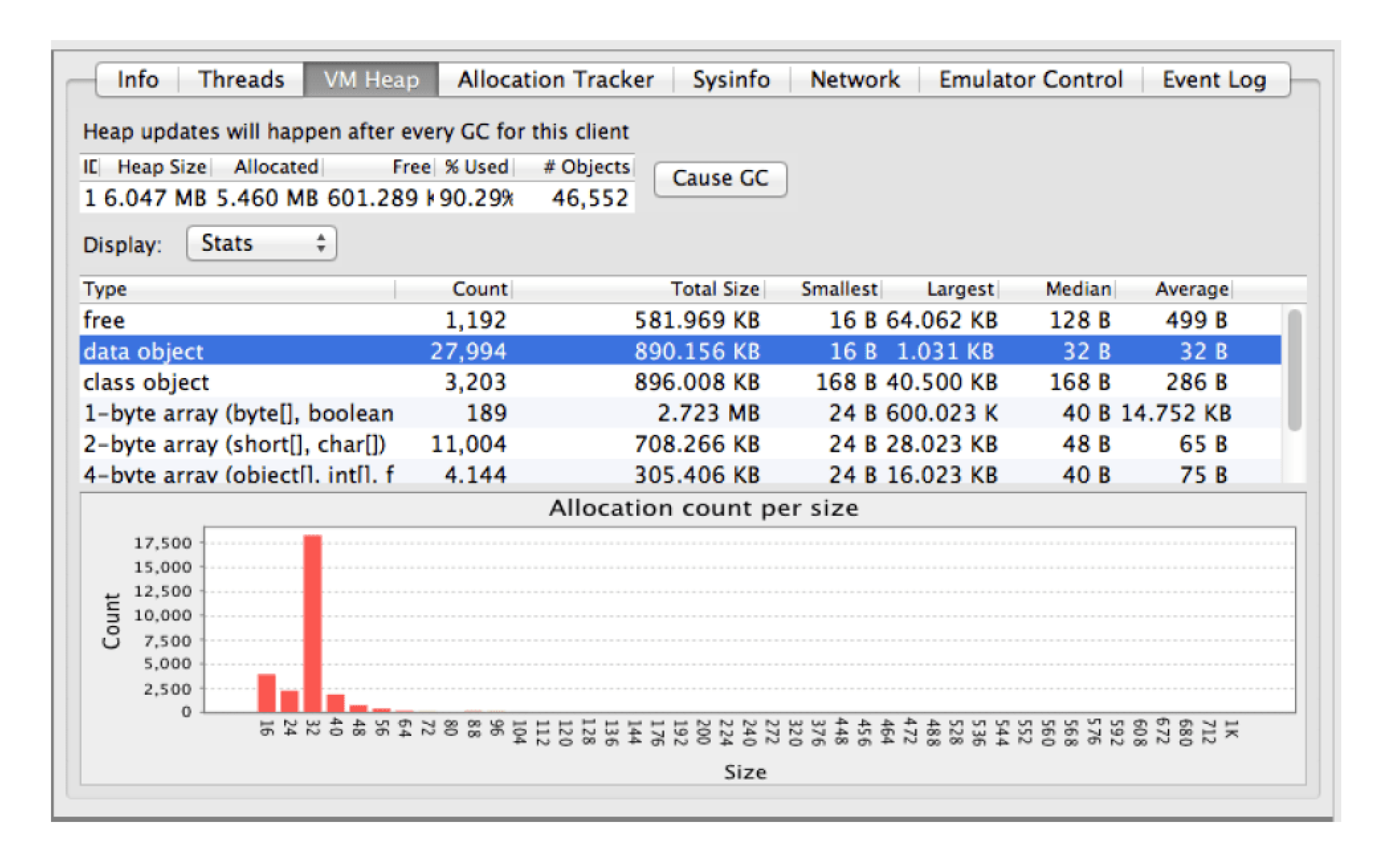


关于 DDMS 的截图、查看进程数、查看日志等这里就不详细说了，有需求的朋友可自行查看官方文档。我想说一下以前经常用的两个功能：VM Heap 和 File Explorer。

先选中一个进程，比如图中的 com.moregg.vida，随后单击左上角的（show heap update）按钮，可以看到进程右边也会显示该图标，表示已经生效。然后单击右边的 VM Heap 标签，可以看到如下图所示的画面。

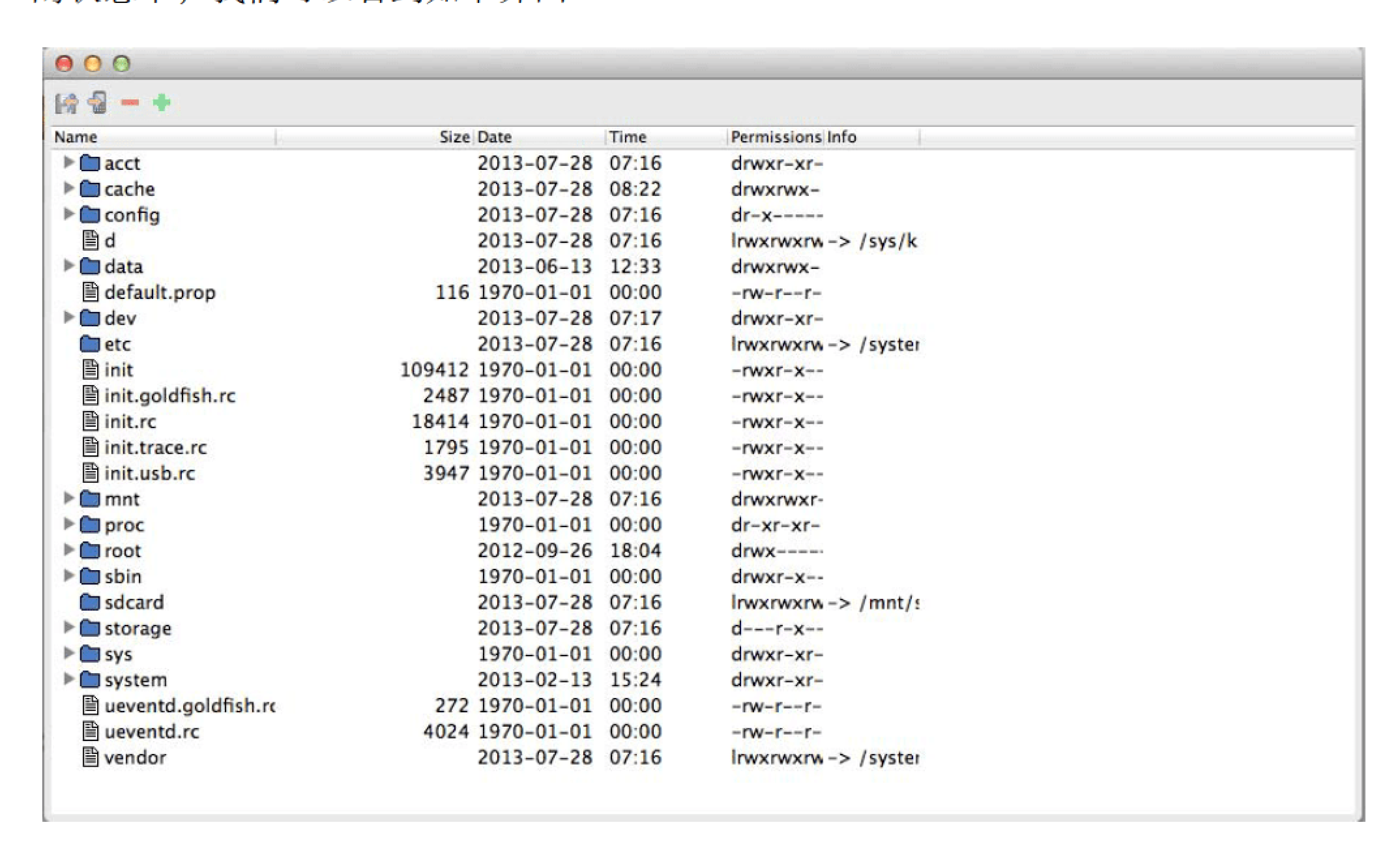


然后单击 Cause GC 按钮来模拟一次 GC 请求，之后，我们可以看到各个内存的使用信息。如下图所示。



执行 GC 请求之后，我们可以继续应用的操作，观察图表中的 data object 这一项，其 Total Size 是否一直会上升。虽然我们的操作会生成更多的对象数据，但是 Java 的 GC 也会不停地进行回收。也就是说，data object 应该稳定在一定范围内，如超过某个值并且越来越大，同时没有下降的趋势，那就是说应用内存必然有泄漏或者没有经过很好地管理。

接下来我们来说说 File Explorer。在 DDMS 常被忽略的菜单中有这一项。在机器 root 的状态下，我们可以看到如下界面。



如果在终端下，相信使用 Linux 的朋友都非常熟悉这个界面了。在这个界面中我们可以将任意的文件 push 到任意目录下（等同于 adb push）或从系统中 pull 出来（等同于 adb pull）。相对来讲，界面化的操作令用户感到非常简便。

Android 的 ADT（Android Developer Tools）很好地和 Eclipse 结合在一起，DDMS 就是其中的一个，其用处也远远不止提到的这些，这就要看每个工程师在工作中的具体需求了。

5.6　Compatibility Test Suite

Compatibility Test Suite 简称 Android CTS，这是一套 Android 原生提供的兼容性测试框架。只有通过 CTS 测试的移动设备才有可能获得 Android 的商标及享受 Android Market 的权限。Android 的 CTS 目的与意义：用户在 Android 系统中有更好的用户体验，并且展示 Android 应用的优越性，使得 Android 开发者更容易编写高质量的 Andorid 程序。

在搭建完 Android 环境之后，如果想要运行 CTS，还需要单独下载一个压缩包来支持运行 CTS 测试。下载地址是：

```
http://source.android.com/compatibility/downloads.html
```



提示：有关 CTS 的安装和使用方法的介绍，在搜索引擎上已经泛滥了，这里就不做过多的讲解。详细可见官方文档：
http://static.googleusercontent.com/external_content/untrusted_dlcp/source.android.com/zh-CN//compatibility/android-4.3-cdd.pdf。

CTS 框架也适合做修改，这样就变成一个能够方便地运行自动化测试用例的平台。

在我个人看来，CTS 不仅是一个测试兼容性的工具，更是一个 Android 自动化测试的代码库。有很多朋友说不知道怎么去写自动化测试用例，找不到比较合适的例子，CTS 就是谷歌给我们的最宝贵的例子。下载源码之后（CTS 源码包括在 Android 系统源码中，在 Git 下载的时候可选择是否包含 CTS），会发现源码中有不计其数的测试代码，是不是热血沸腾了？这里我就给大家举其中一个很简单很常见的例子，它是基于 Android 的 Instrumentation 原生测试框架编写的。关于 Instrumentation 框架会在下一章“常见框架”中有详细说明。我们先来看一下 CTS 的 View_LayoutPositionTest 类，核心代码如下：

```
public class View_LayoutPositionTest
```

```
        extends ActivityInstrumentationTestCase2<ViewLayoutPositionTestStu
bActivity> {

    private Activity mActivity;

    public View_LayoutPositionTest() {
        super("com.android.cts.stub", ViewLayoutPositionTestStubActivity.c
lass);
    }

    @Override
    protected void setUp() throws Exception {
        super.setUp();
        mActivity = getActivity();
    }

    @UiThreadTest
    public void testPositionInParent() {
        View parent = mActivity.findViewById(R.id.testparent);
        View view = mActivity.findViewById(R.id.testview);
        int [] pLocation = new int[2];
        int [] vLocation = new int[2];
        Rect pRect = new Rect();
        Rect vRect = new Rect();

        // baseline not support in view
        assertEquals(-1, view.getBaseline());

        parent.getLocationOnScreen(pLocation);
```

```
    view.getLocationOnScreen(vLocation);

    parent.getDrawingRect(pRect);
    view.getDrawingRect(vRect);

    int left = vLocation[0] - pLocation[0];
    int top = vLocation[1] - pLocation[1];
    int right = left + vRect.width();
    int bottom = top + vRect.height();

    assertEquals(left, view.getLeft());
    assertEquals(top, view.getTop());
    assertEquals(right, view.getRight());
    assertEquals(bottom, view.getBottom());
```

这段代码是非常典型的使用 Instrumentation 框架的例子。先通过 findViewById 来获取被测应用 Activity 中的两个元素：

```
View parent = mActivity.findViewById(R.id.testparent);
View view = mActivity.findViewById(R.id.testview);
```

随后通过 getLocationOnScreen (int[] location)方法来获取 x、y 坐标，并存入数组。

public void **getLocationOnScreen** (int[] location)　　Added in API level 1

Computes the coordinates of this view on the screen. The argument must be an array of two integers. After the method returns, the array contains the x and y location in that order.

Parameters

location　an array of two integers in which to hold the coordinates

最后通过断言来确认 Activity 中的 View 控件离开屏幕的距离是否正确。

```
    assertEquals(left, view.getLeft());
```

```
    assertEquals(top, view.getTop());
```

相信上面的例子对大家来讲非常简单，事实上 CTS 还有很多复杂的例子，并且涵盖了 Android 系统的各个模块，非常值得借鉴。我建议大家多去看看这些例子是怎么写的，这可以很好地帮助自己编写自动化测试。

5.7 Tcpdump/WireShark

有一段时间我和 PPTV 合作做 Android 系统的智能机顶盒，在整个产品中有一个最重要、最核心的功能——视频播放。可是，视频播放往往会出现播放不流畅、分辨率不高等问题，所以需要定位网络问题，比如丢包率，于是就找到了 WireShark 这个工具。

在使用的时候需要两个工具：Tcpdump 和 WireShark。Tcpdump 的下载地址：http://www.tcpdump.org/。

WireShark 的下载地址：http://www.wireshark.org/。

这里用模拟器作为例子，因为该工具需要测试机 root 之后才有权限使用。我们需要将 Tcpdump 放入到 Android 系统的目录中去，命令如下：

```
adb push <tcpdump path> /data/local/tcpdump
```

接着需要分配权限给存有 Tcpdump 文件的目录：

```
adb shell chmod 6755 /data/local/tcpdump
```

然后使用 adb shell 进入类似于 Linux 的终端模式，输入以下命令（其中.pcap 文件保存路径可任意指定）：

```
/data/local/tcpdump -p -vv -s 0 -w /data/local/capture.pcap
```

成功开始抓包之后会出现如下反馈：

```
tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 6553
5 bytes
Got 32
```

这里 Got 的数据即是抓包的数据，会随着时间的延长而不停地增长。在执行抓包的同时可做其他相关的操作，使得数据更精准和集中，停止之后可使用以下命令或通过 DDMS 将文件从 Android 系统中导出（命令中最后的点代表将文件导出到当前目录，也可直接指定路径）：

```
adb pull /data/local/capture.pcap .
```

拿到.pcap 文件之后，需要使用 WireShark 程序来打开，可以看到非常详细的信息，包括每一秒的网络耗时，网络协议，IP 地址等等。

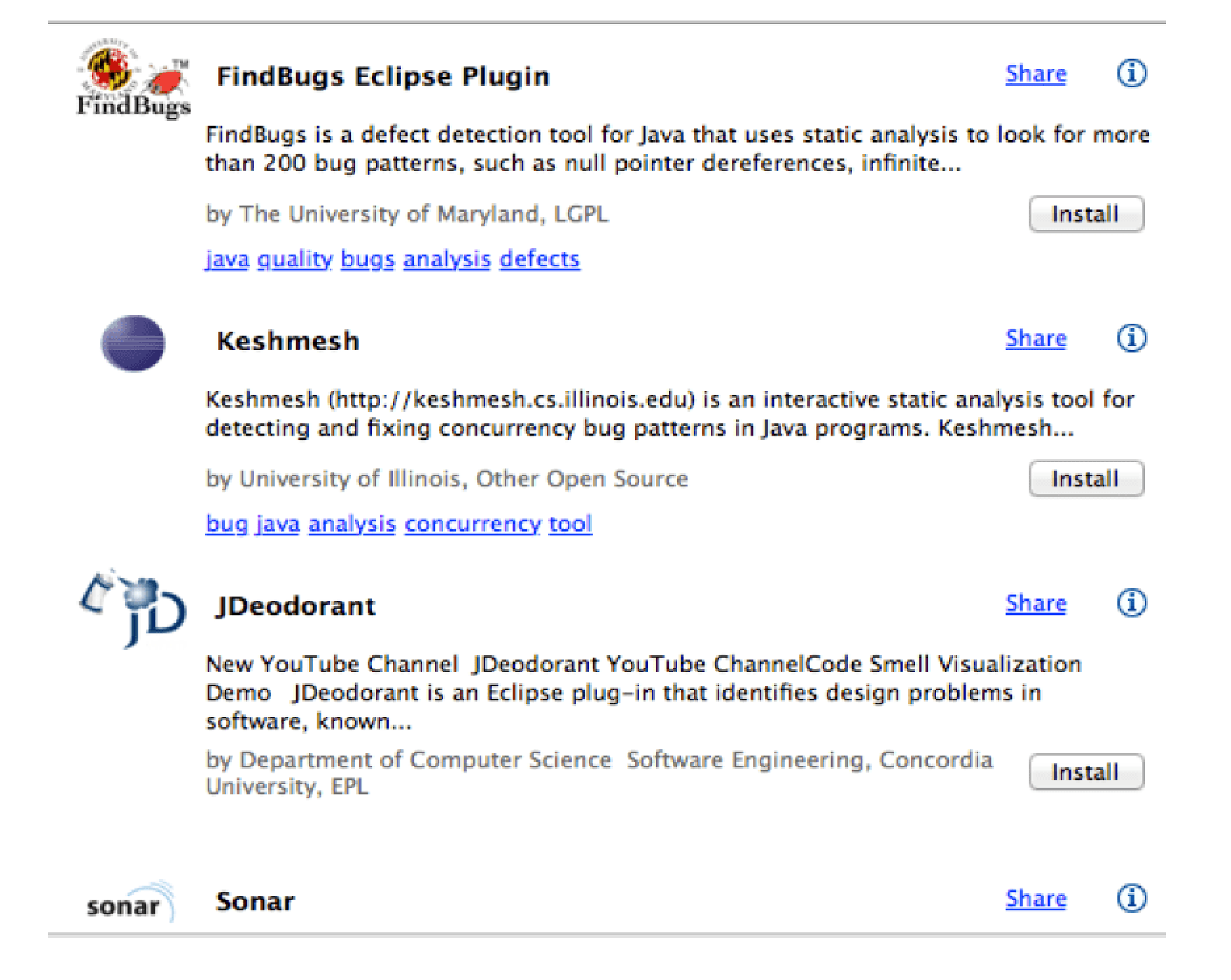


提示：关于 WireShark 日志的详细分析我就不在这里做介绍了，有兴趣的同学可以先熟读 TCP/IP 协议卷一到卷三，同时可参照 WireShark 官方网站的文档，相信帮助会非常大。

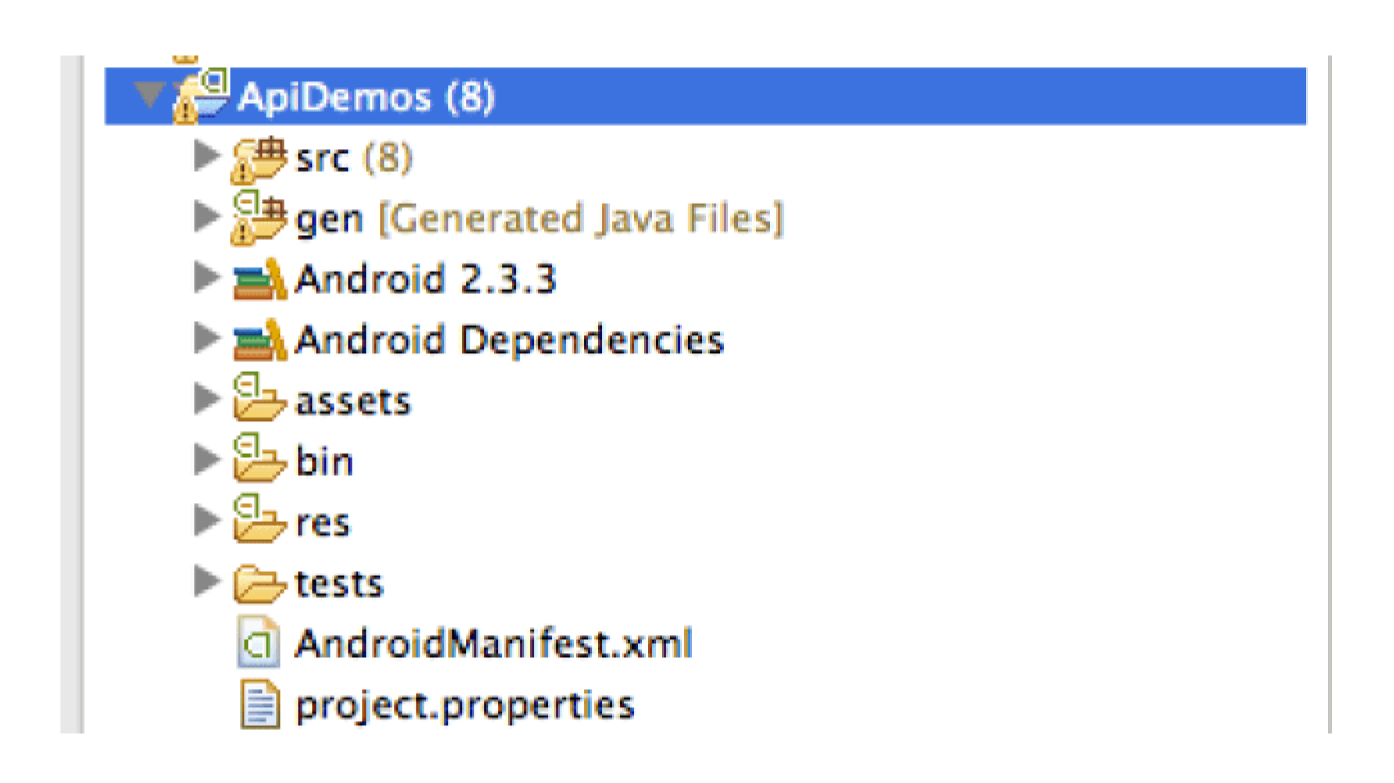
5.8 FindBugs

FindBugs 是一款 Java 静态代码分析工具，与其他静态分析工具不同，FindBugs 不注重样式或者格式，而专注于寻找真正的缺陷或者潜在的性能问题，它可以帮助 Java 工程师提高代码质量以及排除隐含的缺陷。有了静态分析工具，就可以在不实际运行程序的情况下对软件进行分析。

FindBugs 本身可以作为 Eclipse 的插件进行下载，在 Eclipse 的 market 中可找到并下载。



下载安装完毕之后，右键 Ecplise 的项目即会见到 FindBugs 相关的选项，单击 FindBugs 之后便会在项目右边出现找到的问题数量。这里以 Android ADT 自带的项目 Api Demo 为例，使用过 FindBugs 之后的结果如下图显示。



在项目中的每个问题会被定位到具体的每个类上。打开相关类之后就可以在设置断点的地方看到 FindBugs 的标记，将光标移上去之后就会看到相关的修改信息，如下图所示。

```
             *
             */
            path = "";
Comparison of String objects using == or != in com.example.android.apis.media.MediaPlayerDemo_Video.playVideo(Integer)
                // Tell the user to provide a media file URL.
                Toast
                        .makeText(
                                MediaPlayerDemo_Video.this,
                                "Please edit MediaPlayerDemo_Video Activity,"
                                        + " and set the path variable to your media file l
```

另外，右键项目选择 Properties 之后可找到 FindBugs 的标签，在该标签中，最重要的是我们可以自定义 FindBugs 的规则，以保证每次工具是根据设置的策略来寻找缺陷，进一步提升我们的效率。

Properties for ApiDemos

type filter text
- Resource
- Android
- Android Lint Preferences
- Builders
- FindBugs
- Java Build Path
- Java Code Style
- Java Compiler
- Java Editor
- Javadoc Location
- Project References
- Refactoring History
- Run/Debug Settings
- Task Repository
- Task Tags
- Validation
- WikiText

FindBugs

Enable project specific settings   Configure Workspace Settings...

Run automatically,   (also on full build),   analysis effort Default

Store issue evaluations in: (cloud disabled)

Reporter Configuration | Filter files | Plugins and misc. Settings | Detector configuration

Disabled detectors will not participate in FindBugs analysis.
'Grayed out' detectors will run, however they will not report any results to the UI.

Show hidden detectors

Detector id	Pattern(s)	Speed	Provider	Category
AppendingToAnObjectOutputStream	IO	fast	FindBugs	Correctness
AtomicityProblem	AT	fast	FindBugs	Multithrea...
BadAppletConstructor	BAC	fast	FindBugs	Correctness
BadResultSetAccess	SQL	fast	FindBugs	Correctness
BadSyntaxForRegularExpression	RE	fast	FindBugs	Correctness
BadUseOfReturnValue	RV	fast	FindBugs	Dodgy code
BadlyOverriddenAdapter	BOA	fast	FindBugs	Correctness
BooleanReturnNull	NP	fast	FindBugs	Bad practice
CallToUnsupportedMethod	Dm	fast	FindBugs	Dodgy code
CheckExpectedWarnings	FB	fast	FindBugs	Correctness
CheckImmutableAnnotation	JCIP	fast	FindBugs	Bad practice
CheckTypeQualifiers	TQ	slow	FindBugs	Correctnes...
CloneIdiom	CN	fast	FindBugs	Bad practice
ComparatorIdiom	Se	fast	FindBugs	Bad practice
ConfusedInheritance	CI	fast	FindBugs	Dodgy code
ConfusionBetweenInheritedAndOuterMe...	IA	moderate	FindBugs	Dodgy code
CrossSiteScripting	HRS|PT|XSS	fast	FindBugs	Security
DefaultEncodingDetector	Dm	fast	FindBugs	Internation...
DoInsideDoPrivileged	DP	fast	FindBugs	Malicious

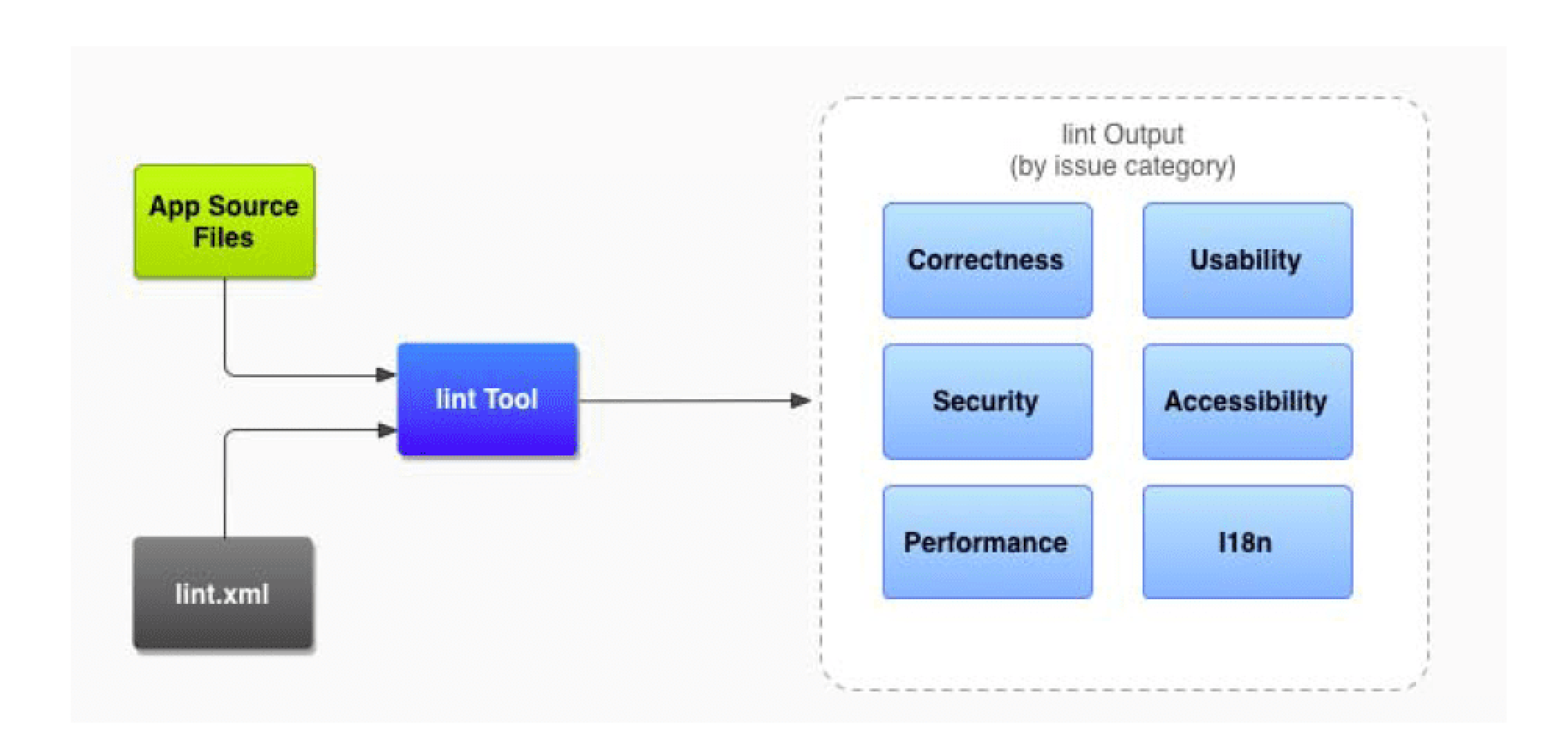
Detector details



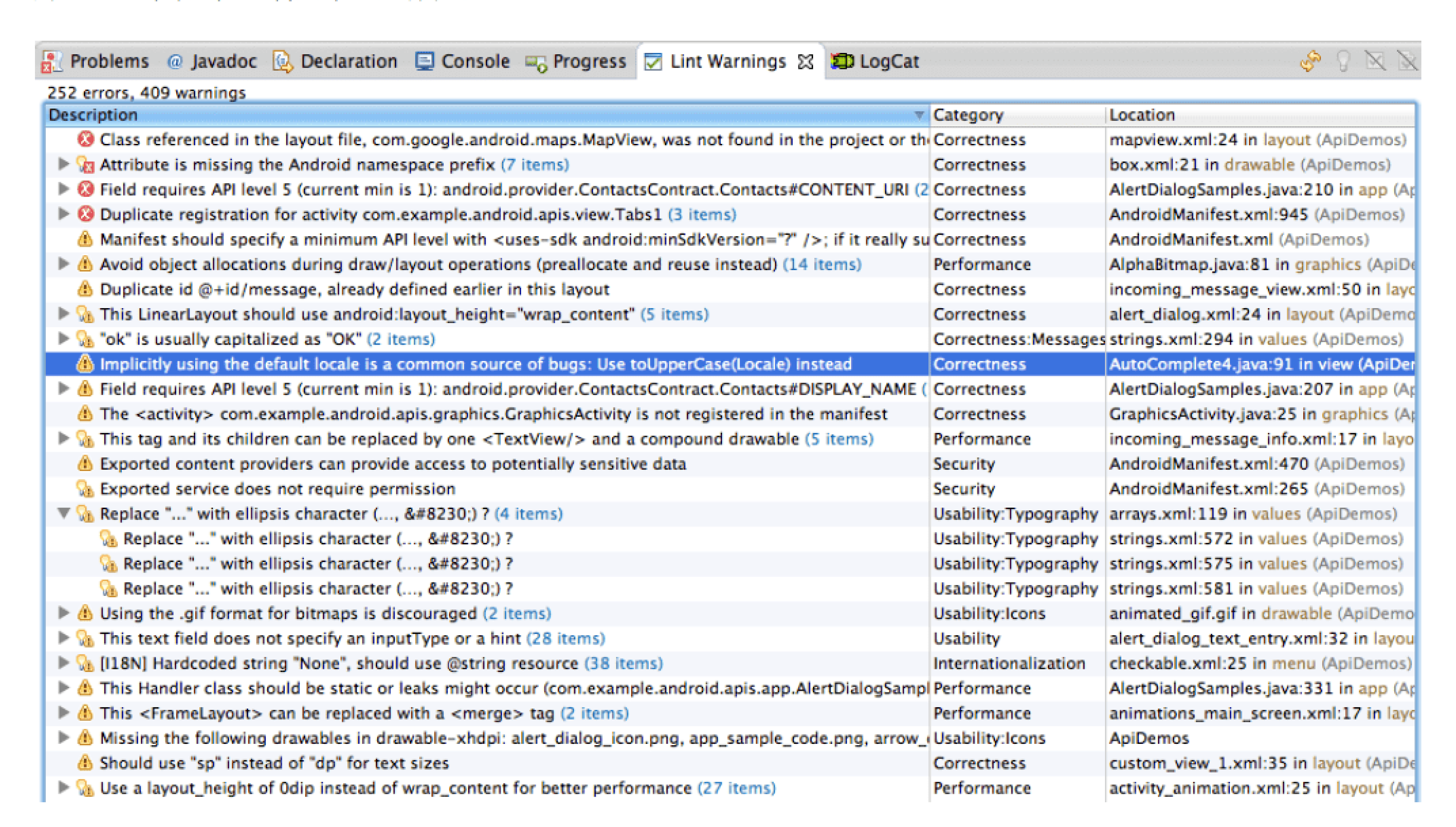
提示： 更多的规则解释描述详见：
http://findbugs.sourceforge.net/findbugs2.html。

5.9 Lint

Lint 和 FindBugs 一样，都是静态代码扫描工具，区别在于它是 Android SDK 提供的，会检查 Android 项目源文件的正确性、安全性、性能、可用性等潜在的 bug 并优化改进。下图简单地描述了 Lint 工具的原理。



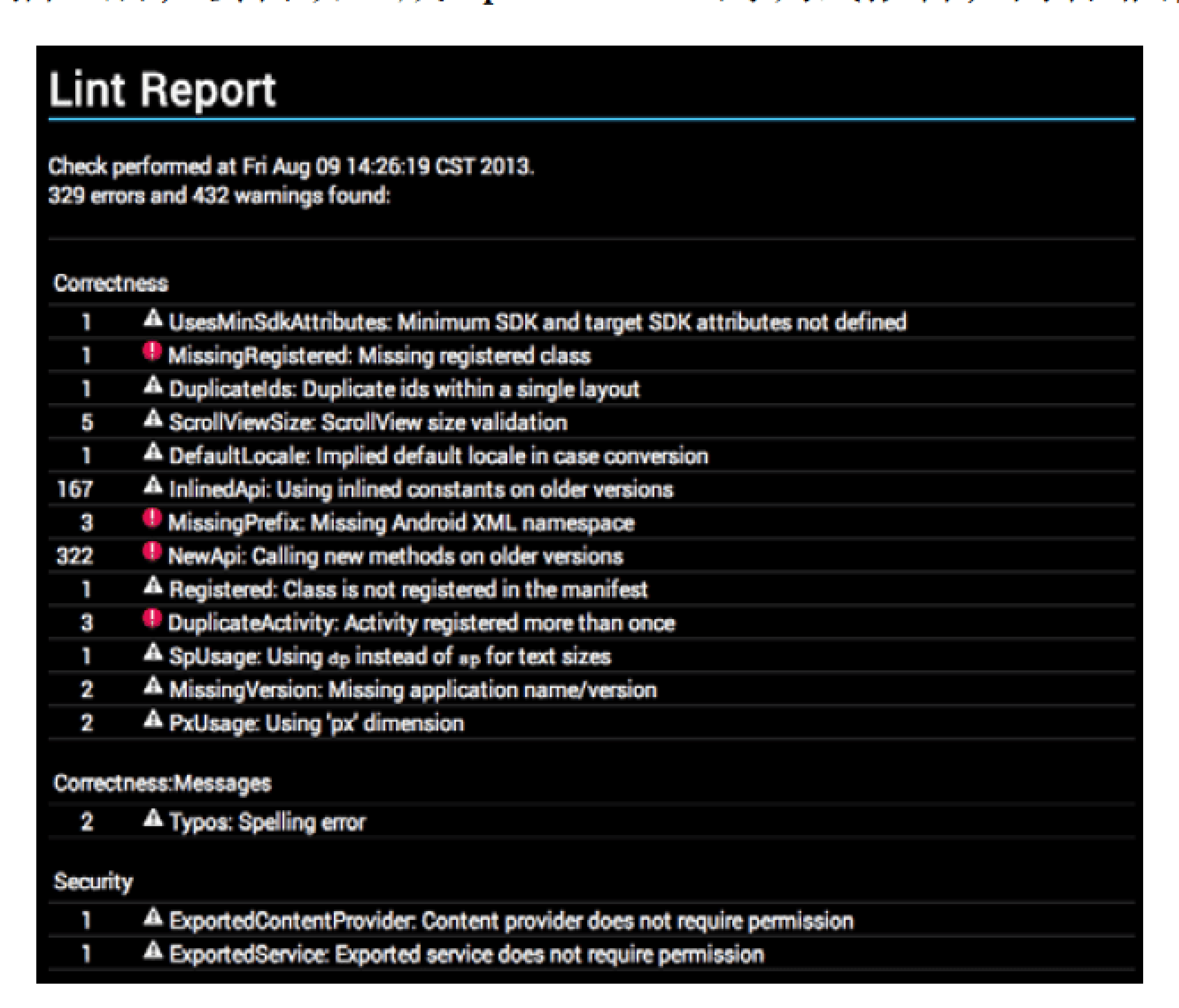
在 Eclipse 中右键工程，在出现的菜单中选择 Android Tools 中的 Run Lint，即可执行 Lint 测试。结果如图所示。



Lint 也可以通过命令行的方式对工程进行测试，并同时生成测试报告。在终端中输入：

```
lint apidemos --html apitest.html
```

代码静态扫描之后，会自动生成 apitest.html 的测试报告，内容非常详细。


Lint Report

Check performed at Fri Aug 09 14:26:19 CST 2013.
329 errors and 432 warnings found:

Correctness

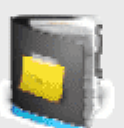
1	UsesMinSdkAttributes: Minimum SDK and target SDK attributes not defined
1	MissingRegistered: Missing registered class
1	DuplicateIds: Duplicate ids within a single layout
5	ScrollViewSize: ScrollView size validation
1	DefaultLocale: Implied default locale in case conversion
167	InlinedApi: Using inlined constants on older versions
3	MissingPrefix: Missing Android XML namespace
322	NewApi: Calling new methods on older versions
1	Registered: Class is not registered in the manifest
3	DuplicateActivity: Activity registered more than once
1	SpUsage: Using dp instead of sp for text sizes
2	MissingVersion: Missing application name/version
2	PxUsage: Using 'px' dimension

Correctness:Messages

2	Typos: Spelling error

Security

1	ExportedContentProvider: Content provider does not require permission
1	ExportedService: Exported service does not require permission


提示： 关于 Lint 更多的参数命令和使用信息可参照：
http://developer.android.com/tools/help/lint.html 和
http://developer.android.com/tools/debugging/improving-w-lint.html。

5.10 反编译、重编译

在 Android 早期做 apk 应用的时候，很多人对于反编译还没有概念，或者说还没有防范意识。随着反编译的流行，为了保证自己的产品不会变成满天飞的山寨应用，更多的企业选择了混淆编译以及将核心逻辑编译成.so 等方法，以防止别人的反编译。由于对移动安全方面了解甚少，所以本书只有这一节涉及到安全方面的皮毛也许，笔者有兴趣的朋友可以翻阅相关资料学习。

第一个需要用到的工具是 Apktools，下载地址：

https://code.google.com/p/android-apktool/downloads/list 。解压之后会得到

apktools.jar 文件，打开终端指向 apktools 根目录，输入以下命令即可完成反编译：

```
java -jar apktool.jar d xxx.apk
```

成功反编译的话可以看到如下日志并生成一个和 apk 名字一样的文件夹。

```
I: Baksmaling...
I: Loading resource table...
I: Loaded.
I: Decoding AndroidManifest.xml with resources...
I: Loading resource table from file: /Users/apple/Library/apktool/framework/1.apk
I: Loaded.
I: Regular manifest package...
I: Decoding file-resources...
I: Decoding values */* XMLs...
I: Done.
I: Copying assets and libs...
```

这样就可以根据我们的需求对 mainfest.xml 等文件做相应的修改了（比如增加权限等）。

接下来我们就需要进行重编译，同样要使用 Apktools 工具。在终端中输入：

```
java -jar apktools.jar b <反编译出来的文件夹>
```

重编译成功的话会在文件夹中看到 build 和 dlist 文件夹，重新编译成功的 apk 就保存在 dlist 文件夹中。但是这个 apk 目前还不能正常安装，原因是该 apk 还是未签名状态（unsign）。接下来我们就说说 Android 应用的签名。

在 Eclipse 的设置中点击 Android 标签中的 Build 选项，可以看到 Eclipse 中 Android 工程默认的签名都是 Android ADT 自带的 debug 签名，如图所示。

同样地，右键一个项目，选择 Android Tools 中的 Expert Signed Application Package，能够成功创建所选签名证书的应用 apk。而需要让一个没有签名的应用程序签上名的话，需要第 3 个工具：re-sign.jar。re-sign-jar 的下载地址：

```
http://troido.com/downloads/1/category
```

使用该工具能将之前重编译的应用变成带有签名证书的应用。

在整个反编译过程中最重要的目的除了修改以外还有一个，那就是查看源代码。这里就需要第 4 个工具 dex2jar，下载地址：

```
https://code.google.com/p/dex2jar/downloads/list
```

解压之后，运行一下命令给予相应的权限：

```
chmod +x dex2jar.sh
```

接着使用该文件进行反编译：

```
./dex2jar.sh xxx.apk
```

成功之后会出现_dex2jar.jar 文件，然后我们需要下载第 5 个工具 JD-GUI，下载地址：http://java.decompiler.free.fr/。使用该工具打开<apk name>_dex2jar.jar 文件，就能够直接看到应用程序的代码，如下图所示。

本节简单介绍了反编译、重编译、签名等需要使用的工具。由于相关的其他测试我用得也不多，所以更多的知识还需要读者多谷歌学习了。

5.11 Ant

Ant 是著名的 Java 开源组织 Apache 的一个项目，是一个基于 Java 的 build 工具。在 Android 项目中可以使用 Ant 来提升工作的效率。比如自动编译、自动打多渠道包等。本节会对 Ant 做一个基础性的介绍。

首先我们要先安装 Ant，下载地址：http://ant.apache.org/bindownload.cgi。安装之后先来尝试自动编译一个 Android 项目。就拿 Android 自带的 Spinner 项目为例，终端定位到 Spinner 项目之后执行以下命令：

```
android update project -n Spinner -t 1 -p /Users/apple/Desktop/worksp
ace/Spinner
```

每个参数具体描述：

-n 项目名称

-t 所选择项目对应的 target ID。可以通过 android list target 来查看不同 Androd 版本所对应的 target ID

-p 选择 build.xml 生成的路径

执行命令之后我们可以看到以下的日志：

```
Updated project.properties
Updated local.properties
Added file /Users/apple/Desktop/workspace/Spinner/build.xml
Added file /Users/apple/Desktop/workspace/Spinner/proguard-project.txt
It seems that there are sub-projects. If you want to update them
please use the --subprojects parameter.
```

同时我们能够在刚刚-p 指定的路径（/Users/apple/Desktop/workspace/Spinner）

中看到有 build.xml 生成。再输入 ant debug，在日志最后部分我们能看到生成应用 apk 以及编译成功等信息，如下面日志中的显示。

```
-do-debug:
[zipalign] Running zip align on final apk...
     [echo] Debug Package: /Users/apple/Desktop/workspace/Spinner/bin/Spi
nner-debug.apk
     [propertyfile] Creating new property file: /Users/apple/Desktop/work
space/Spinner/bin/build.prop
     [propertyfile] Updating property file: /Users/apple/Desktop/workspac
e/Spinner/bin/build.prop
     [propertyfile] Updating property file: /Users/apple/Desktop/workspac
e/Spinner/bin/build.prop
     [propertyfile] Updating property file: /Users/apple/Desktop/workspac
e/Spinner/bin/build.prop


     -post-build:


     debug:


     BUILD SUCCESSFUL
     Total time: 12 seconds
```

我们可以根据日志上显示的路径，找到已经编译完成的 debug 的应用 apk。在做代码覆盖率的时候也需要用到 Ant，Ant 的 build.xml 还能够配置很多参数，用来做各种自动打包的工作，在这里就不详细阐述了，有兴趣的朋友可以根据自己遇到的具体问题谷歌相关解决方案。

5.12 Charles

Charles 是一个能够对 iOS 设备进行远程 http 和 https 抓包的应用。这个工具使用起来能够让人心潮澎湃。接下来我们一起了解一下这个工具吧。

首先是 Charles 的下载链接：

http://www.charlesproxy.com/latest-release/download.do。下载安装并打开之后，可以看到如下界面，这个工具的启动界面还是很漂亮的。



然后我们需要打开 iOS 设备的 wifi 设置，点击底部的手动标签，填入安装 Charles 的 Mac 机器的 ip 地址和端口号，如下图所示：

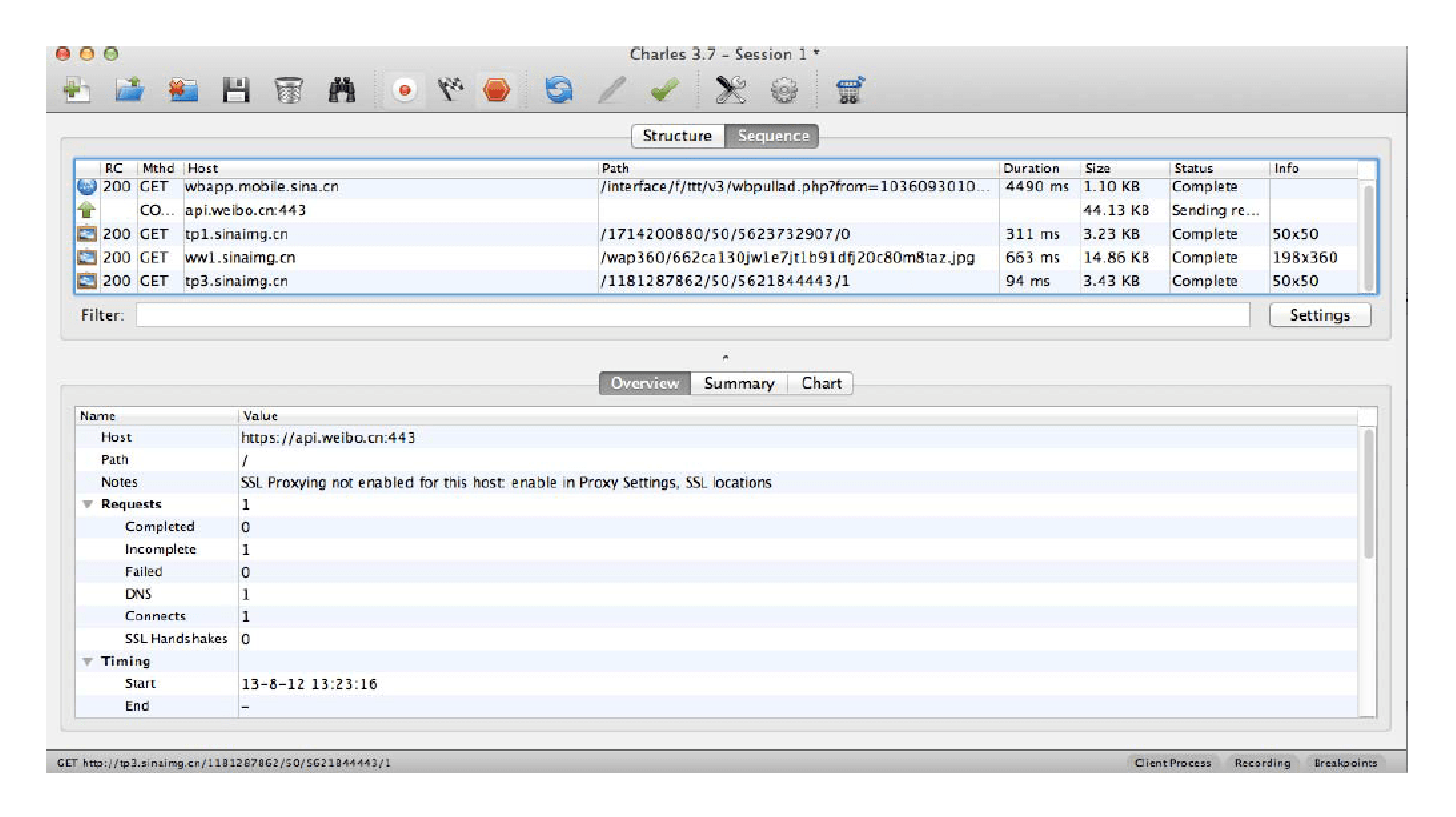


设置完毕之后，打开一个应用（这里以新浪微博为例子），会看到已经启动的 Charles

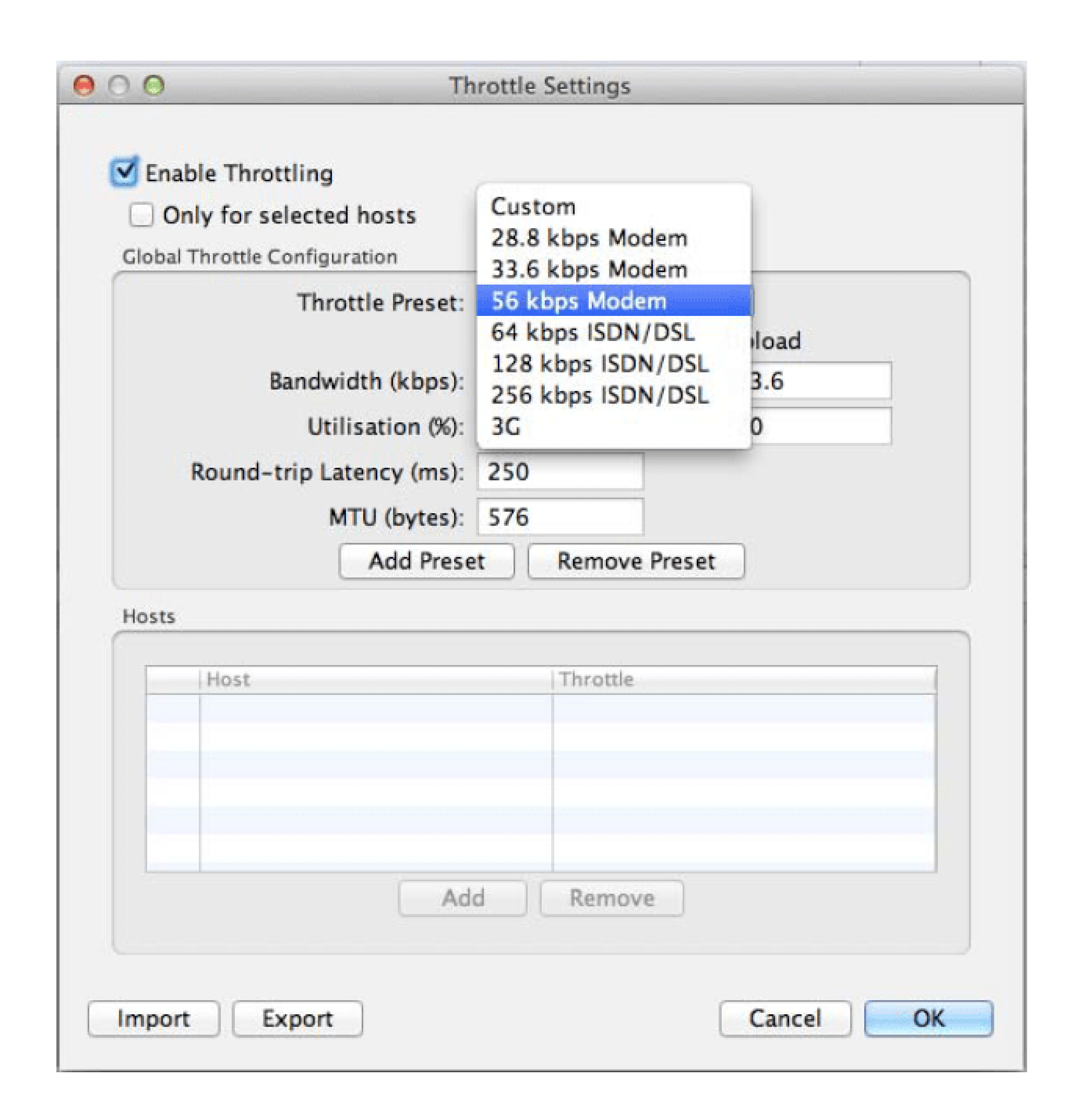
会出现一个对话框，如下图显示。



我们选择允许之后，会出现非常详细的结构以及每次请求的详情。这的确是令人热血沸腾的事情。



在测试移动应用的过程中，很多测试工程师都会遇到这样的测试场景——模拟各种速度的网络，Charles 可以支持网络的模拟。在打开 Charles 之后，可以看到在 Proxy 选项下有 Throttle Settings，点击之后选择 Enable Throttling 就可看到如下界面：



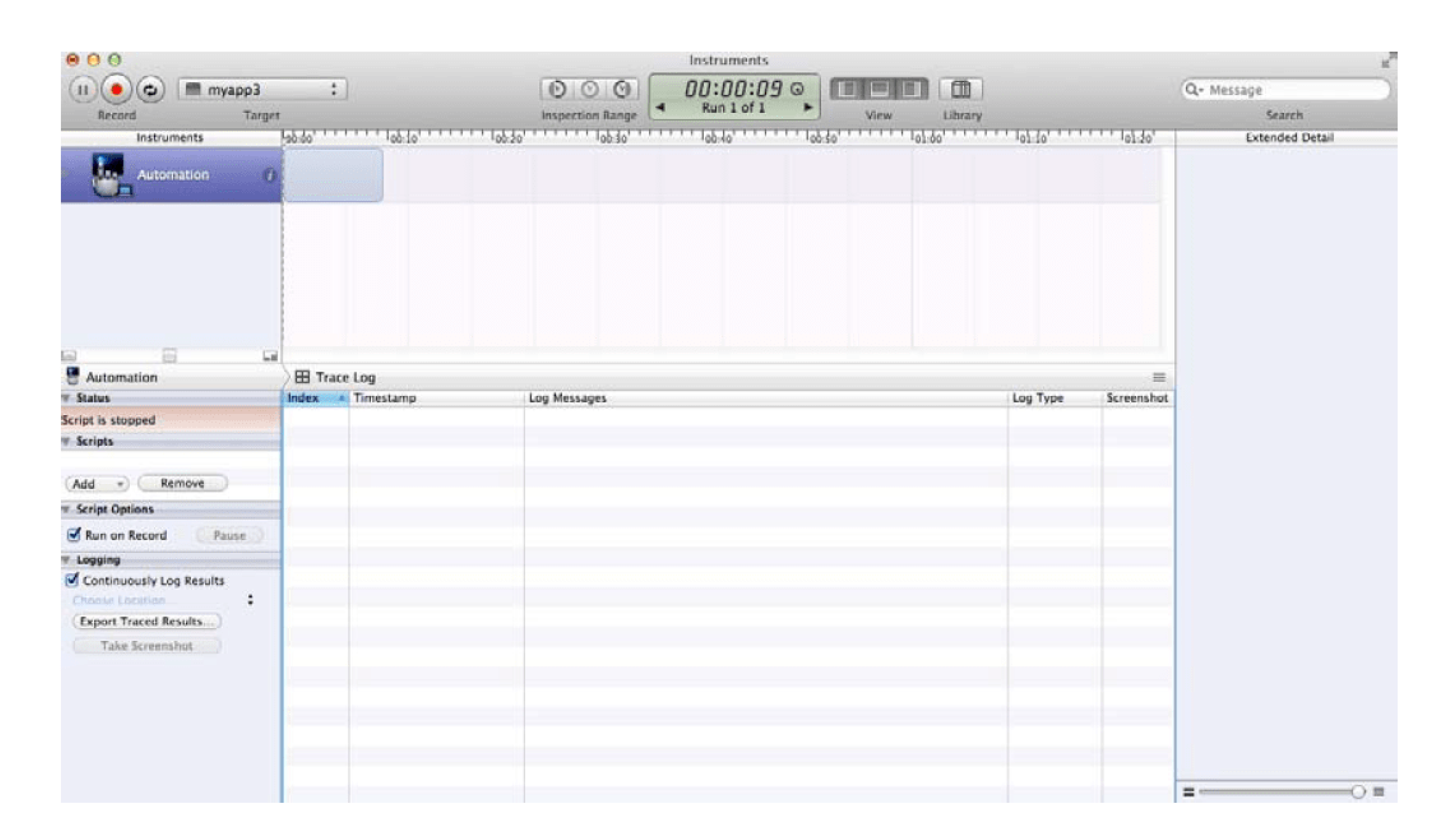
到这里就不用我详细说了吧，根据自己的需求进行相关设置之后就可以模拟不同速度的网络了。

不得不说，由于 iOS 系统权限管理得比较紧，所以测试工程师想要拿到一点数据都困难。相信 Charles 在抓包方面会给广大 iOS 测试工程师带来很大的帮助。

5.13 Instruments

很多测试行业朋友会问怎么测试 iOS 的应用。我觉得无论怎么说，如果要正式进行 iOS 应用的开发，mac 就是必需品。任何模拟器或黑苹果都会出现意料之外的事情来阻扰测试活动，得不偿失。

这里先来介绍 mac xcode 自带的测试工具 Instruments 中的 Automation 这个功能。首先打开 Instruments，如下图所示。



Automation 支持用 JavaScript 编写脚本来进行 iOS 设备的界面自动化测试。这里我给出 iOS 系统下模拟 Android Monkey 测试工具的核心代码：

```
//设置屏幕宽度和高度、同时获取当前窗口
var iphone_Width = 320;
var iphone_Height = 480;
UIALogger.logMessage("开始随机屏幕事件测试");
UIATarget.localTarget().delay(1);
window = UIATarget.localTarget().frontMostApp().mainWindow();

// 随机生成一个坐标点
function randomPoint()
{
    var ret = new Object();
    ret.x = Math.ceil(Math.random()*iphone_Width);
```

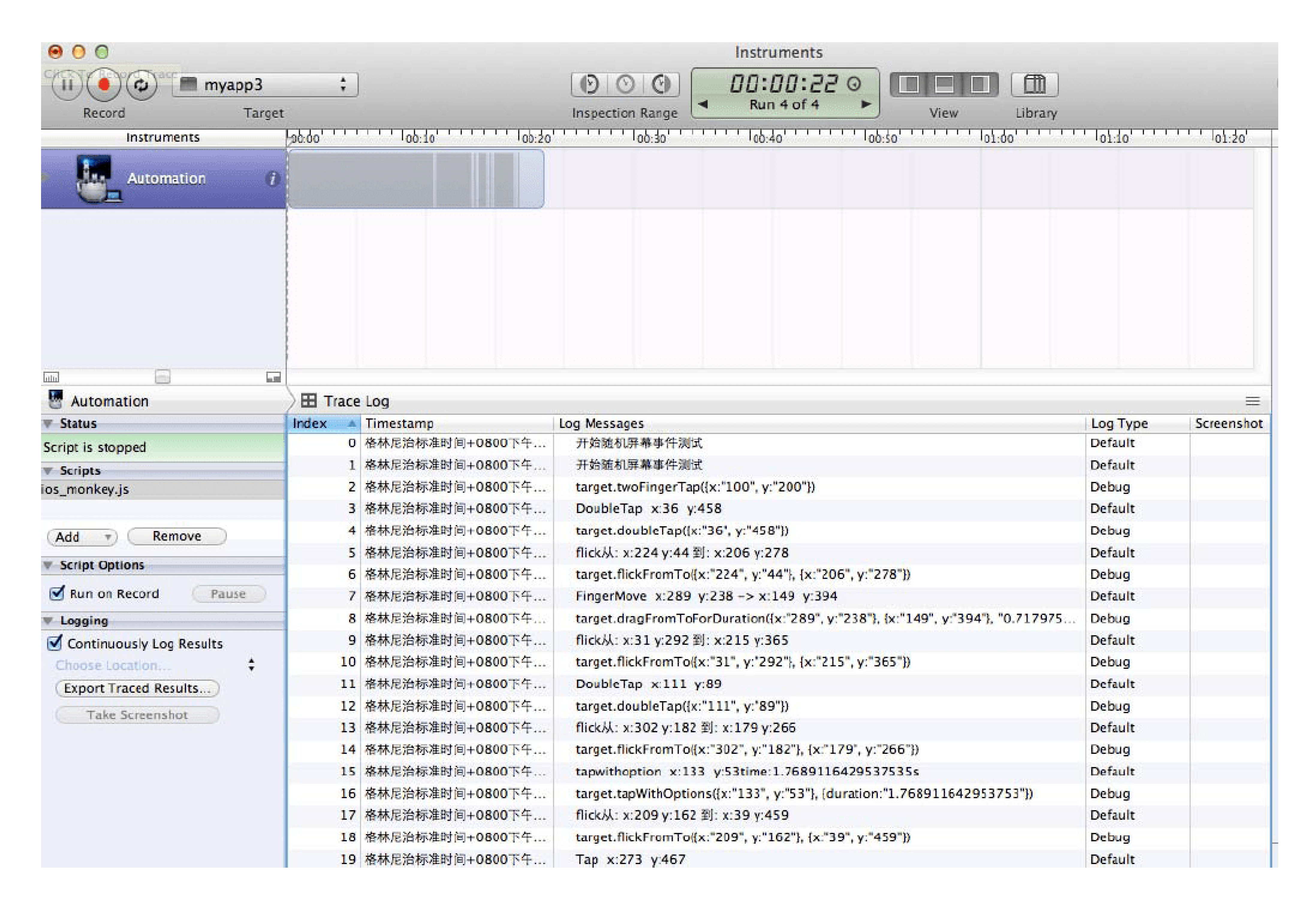
```
        ret.y = 480-Math.ceil(Math.random()*iphone_Height);
        return ret;
    }



    // 单击屏幕
    function Touch()
    {
        var pt = randomPoint();
        UIALogger.logMessage("Tap  x:"+pt.x+"  y:"+pt.y);
        UIATarget.localTarget().tap(pt);
    }

    // 双击屏幕
    function DoubleClick()
    {
        var pt = randomPoint();
        UIALogger.logMessage("DoubleTap  x:"+pt.x+"  y:"+pt.y);
        UIATarget.localTarget().doubleTap(pt);
    }
```

脚本编写完之后，保存成为 Monkey.js 文件。在 Automaiton 的界面的 Scripts 中添加上 Monkey.js 文件，同时选择要执行的应用和设备，最后点击 Record，即可在设备上执行 Monkey 测试了。具体如下图所示：



以上仅仅是实现了应用的随机点击功能，大家可以根据自己的需求再添加滑动、多点触控等方法。当然 Instruments 还提供了用于方便地检查内存泄漏、代码静态分析等工具，这里就不多做介绍了。在 apple 的官方文档中都有详细的介绍，在这里我就不浪费大家的时间了。更多详细内容可参考：http://developer.apple.com/library/iOS/documentation/DeveloperTools/Conceptual/InstrumentsUserGuide/Introduction/Introduction.html。

5.14　小　结

本章对日常会用到的测试工具做了一定的介绍和实践分析，希望对广大读者能有所帮助。本书的定位不是纯技术书籍，所以没有对工具进行全面和深入地讲解，但是希望能够达到抛砖引玉的效果。

读书笔记



第 6 章　常用框架介绍和实践

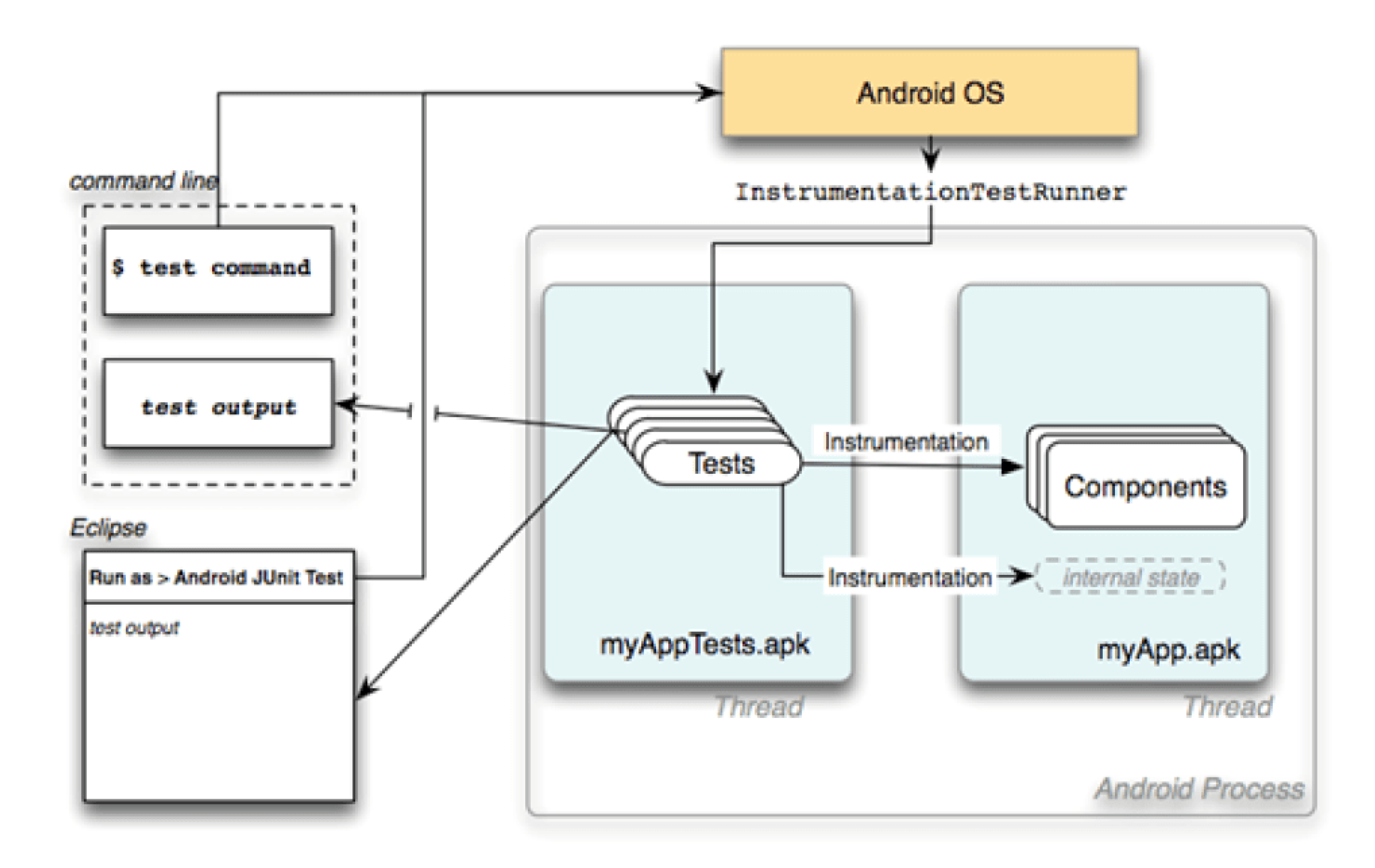
上一章我们对移动互联网客户端常用的测试工具做了介绍，虽然框架从广义上来讲也是工具的一种，但为了让读者各取所需，专门为框架另开一章。本章会对常用框架做一定的介绍和实践分析，如果读者对其他框架感兴趣的话，可自行搜索，这里就不做特别介绍了。

6.1 Instrumentation

Android 提供了一系列强大的测试工具，针对 Android 的环境，扩展了业内标准的 JUnit 测试框架。尽管可以使用 JUnit 测试 Android 工程，但 Android 工具允许我们对应用程序的各个方面进行更为复杂的测试，包括单元层面及框架层面。

Android 执行测试活动的核心就是 Instrumentation 框架，在该框架下我们可以实现界面化测试、功能测试、接口测试甚至单元测试。Instrumentation 框架通过在同一个进程运行主程序和测试程序来实现这些功能。

从整体框架来看，Instrumentation 和 JUnit 测试的框架有很多相同之处，测试环境结构如图所示。



在 Android 系统中，测试程序也是 Android 程序，因此，它和被测试程序的构建方式有很多相同的地方。SDK 工具能帮助用户同时创建主程序工程及它的测试工程。可以通过 Eclipse 的 ADT 插件或者命令行来运行 Android 测试工具。Eclipse 的 ADT 提供

了大量的工具来创建测试用例，运行以及查看结果。更多有关 Instrumentation 的资料可查询：http://developer.android.com/reference/android/app/Instrumentation.html 。

在 Instrumentation 框架中，被继承使用最多的类是 ActivityInstrumentationTestCase2，在 Android 自带的例子中可以很清楚地看到该类，或者说 Android junit test 具体的试用方法，这里就不再多说了。众所周知，robotium、athrun 等框架都是基于 Instrumentation 的，也都很火，它们都将调用接口封装成了日常的操作，使得测试工程师能够更方便地编写测试用例。但也有部分测试工程师觉得它们并不是特别好用，这就会产生一个疑问，真的有很多公司使用这类测试工具在实际项目中自动化吗？接下来，我们主要就围绕这个问题说一说。

先说 Instrumentation。在上文已经提到过该框架几乎能够做你想得到的所有类型的测试，那么，基于该框架的、我们熟知的其他框架是不是也是如此呢？答案是“是”。不要以为 robotium 只是一个 Android 上模拟用户操作的功能自动化框架，它也许是、也许不是，框架是死的，你是活的。

我们再来说说为什么有的测试人员觉得 robotium 等框架不适合使用，有以下 3 种原因（当然，如果你也觉得自己有以下情况，那么要提高警惕，否则最终将一事无成）：

（1）自己根本没有尝试过，觉得自动化根本做不起来，觉得没有用。

（2）项目迭代周期太快，界面变化太快（一般这个原因较多）。

（3）产品本身过于复杂，界面自动化投入产出会非常悬殊。



提示：原因（1）的同学不要放弃治疗，我就不在这里评论了。对于原因（2）和（3）来说，的确大部分的移动互联网企业存在这样的问题，所以在这点上我们只能是具体问题具体分析。“遇神杀神，遇佛杀佛”，关键不在于我们杀谁，而在于我们是否只有 robotium 这个唯一的武器。我用过国内外很多应用，不得不承认，应用与应用之间的差距真的比人与猪之间的差距都大，面对不同的测试对象我们需要随机应变，而不是放弃治疗，你说呢？

当然在我们使用 Android 测试框架的时候也需要掌握一些技巧，不能直接根据 Android 工程例子中的 Spinner 或者 robotium 的 NotePad 的例子，上来就画葫芦，那样的话，尽管你能够画得出几个倭瓜，但还是不知道怎么放入到项目使用，最终回归到放弃治疗状态。

6.1.1 技巧一

在模拟用户操作之前，我们需要先写出整个测试环境是否符合用例执行要求的测试用例，比如如下显示的应用，虽然一般应用都会比这个例子复杂很多，但思路都是一样的。



在我们的测试自动化跑起来之前，首先肯定是准备测试数据，启动被测应用，然后再开始做和功能业务相关的测试。在这之前，我们可以先写些测试用例以确保之后的用例可以被正常执行。如上图所示 4 个控件对象的定义如下：

```
Activity mActivity;
EditText mEditText;
RadioButton mTest1;
RadioButton mTest2;
```

我们首先应该写的用例如下：

```
public final void testActitvity() {
        assertNotNull(mActivity);
```

```
    }

    public final void testInputs() {

            assertNotNull(mEditText);
    }

    public final void testRadiobuttonchecked() {

            assertTrue(mTest1.isChecked());
            assertFalse(mTest2.isChecked());

    }

    public final void testFieldStartempty(){
            assertNull(mEditText.getText().toString());

    }

    public final void testWindowsOnScreen(){
            final Window window = mActivity.getWindow();

            final View  og = window.getDecorView();
            ViewAsserts.assertOnScreen(og, mEditText);
        ViewAsserts.assertOnScreen(og, mTest1);
        ViewAsserts.assertOnScreen(og, mTest2);

    }
```

在开始执行正式用例之前，我们先要确认 Activity 是否启动，必要的控件是否都显示出来了，等等。

6.1.2 技巧二

说到这里就要讲技巧二了——ViewAsserts 类，如下图所示。详细内容参见：

http://developer.android.com/reference/android/test/ViewAsserts.html

public class

ViewAsserts

extends Object

Summary: Methods | Inherited Methods | [Expand All]

Added in API level 1

java.lang.Object
↳android.test.ViewAsserts

Class Overview

Some useful assertions about views.

Summary

Public Methods	
static void	assertBaselineAligned (View first, View second) Assert that two views are aligned on their baseline, that is that their baselines are on the same y location.
static void	assertBottomAligned (View first, View second) Assert that two views are bottom aligned, that is that their bottom edges are on the same y location.
static void	assertBottomAligned (View first, View second, int margin) Assert that two views are bottom aligned, that is that their bottom edges are on the same y location, with respect to the specified margin.

ViewAsserts 的测试方法都是静态的方法。我们能够在 Android 官方文档中看到该类还提供了很多控件之间关系以及控件所在位置定位验证的方法。一般控件都会在.xml 或者.java 文件中清楚地定义位置或者和其他控件的关系，通过使用这些方法，能对应用的各个界面上所使用的控件做全方位的测试。



提示：我个人觉得，如果 Android 应用要用 Instrumentation 做自动化测试的话，从投入产出比的角度来讲，最先该做的是每个界面的启动、截图、网络状态、控件显示，控件位置、xml 文件的参数定义（如果是多语言化的应用更应该做这个测试了）的测试。但现在我看到更多的是，很多测试工程师先急着模拟用户的操作，急着将 Android junit Test 跑起来。我想多嘴一句，不要忘记了测试自己产品的初衷，测试这个职业注定是要求我们用最少的时间将做尽可能多的事。所以在做一件事情之前要三思而后行，问问自己测试的目的是什么？是不是还有更重要事情可以做？而不是一味地去想着一口气吃成个胖子。

6.1.3 技巧三

其实这个真的算不上什么技巧，只是在自动化脚本集成的时候比较有效。在自动化测试最终集成的时候，Instrumentation 担任的角色可能只是整个过程中的一小部分。比如我现在负责做移动应用的自动化测试搭建，其中有一项就是多分辨率的自动化测试。以下是我在 Android 上从开始思考到最终完成任务的两套思路：

（1）以 Python 作为自动化测试方案的主导。python 的 os 模块用来调用 adb，负责启动模拟器和关闭模拟器；adb 负责安装和卸载应用；MonkeyRunner 负责模拟应用必须的操作（比如登录等）；MonkeyRunner 负责截图以及对比最终结果。

（2）以 Python 作为自动化测试方案的主导。Python 的 os 模块用来调用 adb，负责启动模拟器和关闭模拟器；adb 负责安装和卸载应用；appium 负责操作；MonkeyRunner 负责截图以及对比最终结果。

不过最终这两套方案都没有被采纳，原因是因为各种“坑”。其实看到这里，我相信有点基础的同学都会想到两点，一是为什么要用模拟器呢？模拟器明显没有真机靠谱；二个是多分辨率的自动化测试，以前自己也想做过，应该没有难点啊。

不采纳真机的原因除了的确没有那么多真机以外，还有一个很重要的原因就是真机存在连接不稳定、网络不稳定、adb 服务不稳定等各种不稳定因素。再有就是我们不能保证真机的 rom 都是 root 过的，很多工具和脚本会受到限制。

觉得没有难点的同学，其实跟我当初想得一样，感觉这个自动化方案如果让我自己去做，根本就是没有任何难度的。但是真理又再次告诉我们，很多事情的确只有在做的过程中才知道其“坑”之多，比如如何判断模拟器完全启动、如何完美地替模拟器开始界面解锁等，其中最坚难的就是各个脚本之间的衔接。也许每个过程看似很简单，但是最终要把它们衔接起来的时候却变得不是那么容易了。

我自己最后采取的方案是，python 作为自动化测试方案的主导，python 调用 shell 脚本负责启动和关闭模拟器；adb 负责安装和卸载应用；MonkeyRunner 负责屏幕的解锁；Instrumentation 负责应用内的操作以及界面控件的验证；MonkeyRunner 负责最终界面的截图和保存；Python 的 PIL 模块负责图片像素级别的对比。其中的 Instrumentation 必须使用命令驱动，而不是像平时直接在 eclipse 中运行的那样。命令具体的信息如下图所示。

Running all tests: adb shell am instrument -w com.android.foo/android.test.InstrumentationTestRunner

Running all small tests: adb shell am instrument -w -e size small com.android.foo/android.test.InstrumentationTestRunner

Running all medium tests: adb shell am instrument -w -e size medium com.android.foo/android.test.InstrumentationTestRunner

Running all large tests: adb shell am instrument -w -e size large com.android.foo/android.test.InstrumentationTestRunner

Filter test run to tests with given annotation: adb shell am instrument -w -e annotation com.android.foo.MyAnnotation com.android.foo/android.test.InstrumentationTestRunner

If used with other options, the resulting test run will contain the union of the two options. e.g. "-e size large -e annotation com.android.foo.MyAnnotation" will run only tests with both the `LargeTest` and "com.android.foo.MyAnnotation" annotations.

Filter test run to tests *without* given annotation: adb shell am instrument -w -e notAnnotation com.android.foo.MyAnnotation com.android.foo/android.test.InstrumentationTestRunner

Running a single testcase: adb shell am instrument -w -e class com.android.foo.FooTest com.android.foo/android.test.InstrumentationTestRunner

Running a single test: adb shell am instrument -w -e class com.android.foo.FooTest#testFoo com.android.foo/android.test.InstrumentationTestRunner

Running multiple tests: adb shell am instrument -w -e class com.android.foo.FooTest,com.android.foo.TooTest com.android.foo/android.test.InstrumentationTestRunner

Running all tests in a java package: adb shell am instrument -w -e package com.android.foo.subpkg com.android.foo/android.test.InstrumentationTestRunner

Including performance tests: adb shell am instrument -w -e perf true com.android.foo/android.test.InstrumentationTestRunner

To debug your tests, set a break point in your code and pass: -e debug true

To run in 'log only' mode -e log true This option will load and iterate through all test classes and methods, but will bypass actual test execution. Useful for quickly obtaining info on the tests to be executed by an instrumentation command.

To generate EMMA code coverage: -e coverage true Note: this requires an emma instrumented build. By default, the code coverage results file will be saved in a /data//coverage.ec file, unless overridden by coverageFile flag (see below)

To specify EMMA code coverage results file path: -e coverageFile /sdcard/myFile.ec
in addition to the other arguments.



提示：更多的信息可以可在：http://developer.android.com/reference/android/test/InstrumentationTestRunner.html 中找到。

6.2　Emma Code Coverage

在软件测试中，测试用例对于被测对象的代码覆盖率而言一直是一个很好的参考标准。Android 中也提供了比较方便的工具 emma。在官方文档中也有对 emma 详细的描述。

emma	true	Runs an EMMA code coverage analysis and writes the output to /data//coverage.ec on the device. To override the file location, use the coverageFile key that is described in the following entry. **Note:** This option requires an EMMA-instrumented build of the test application, which you can generate with the coverage target.
coverageFile	<filename>	Overrides the default location of the EMMA coverage file on the device. Specify this value as a path and filename in UNIX format. The default filename is described in the entry for the emma key.

更多的详情可参考：

http://developer.android.com/tools/testing/testing_otheride.html

emma 可以让我们很方便地统计 Android junit test 的测试代码覆盖率，只要稍做修改，就能帮助我们实现以黑盒手动测试的方法来统计代码覆盖率。

接下来我们就来看一个例子，这个例子使用 Iinstrumentation 框架并结合第 5 章讲到的 Ant 工具实现。

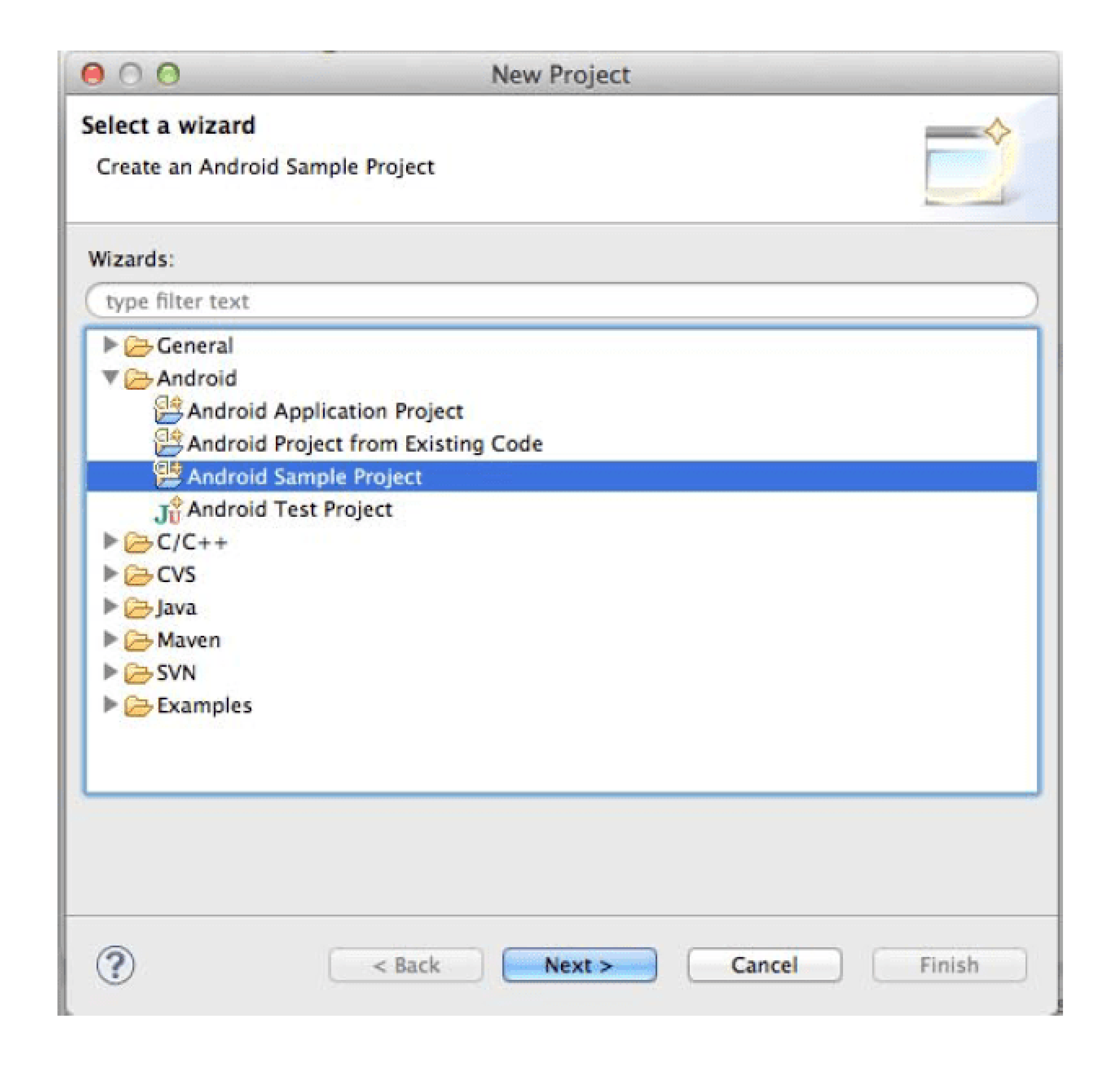
（1）我们先使用如下的 emulator 命令新建一个模拟器：

```
android create avd -n chenye -t 17
-n emulator name
-t android target ID
```

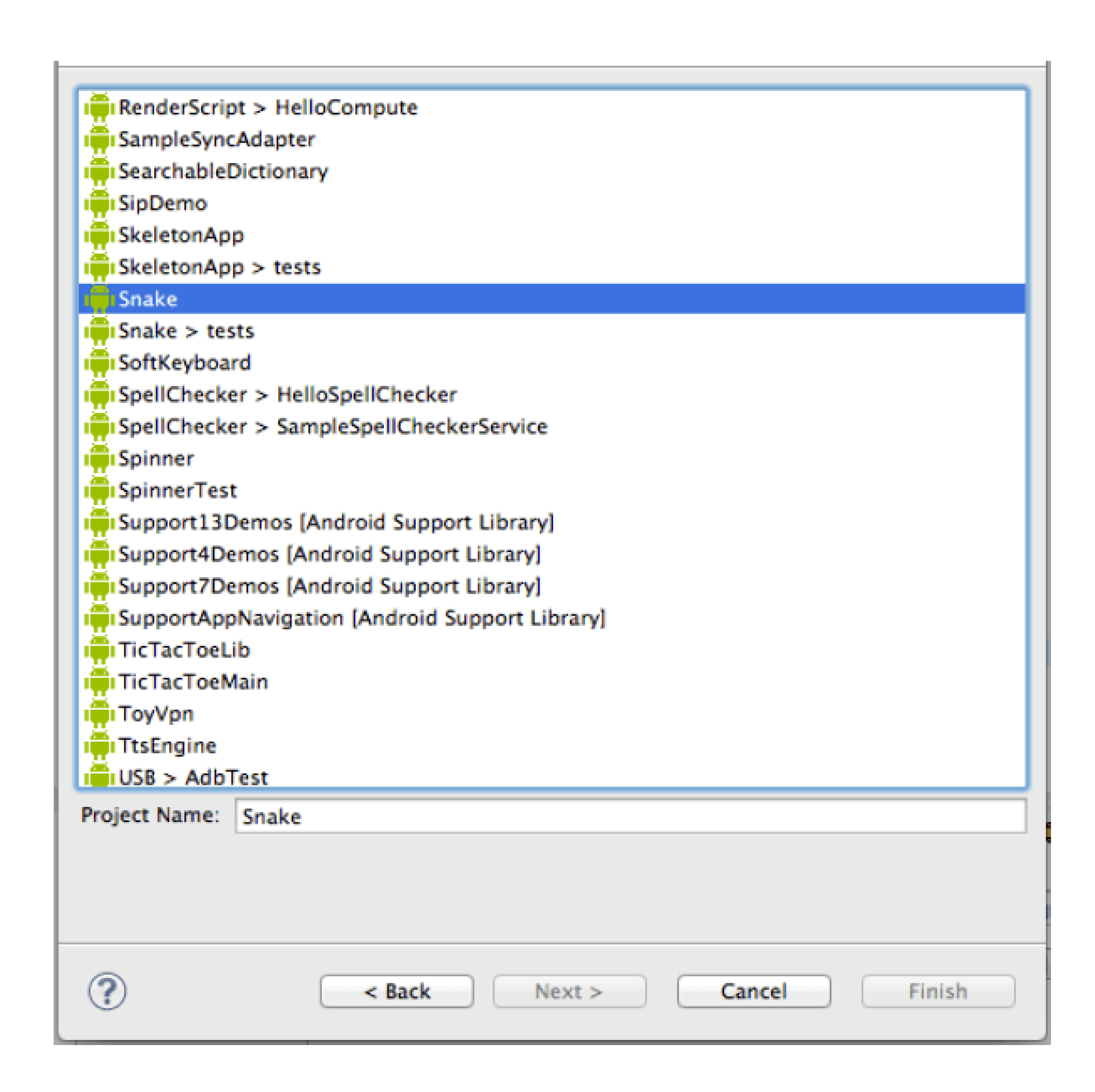
（2）Android APIs 17 对应的版本是 4.1.2，成功建立模拟器之后，可以看到以下日志:

```
Created AVD 'chenye' based on Google APIs (Google Inc.), ARM (armeabi-v7a)
 processor,
with the following hardware config:
hw.lcd.density=240
vm.heapSize=48
hw.ramSize=512
```

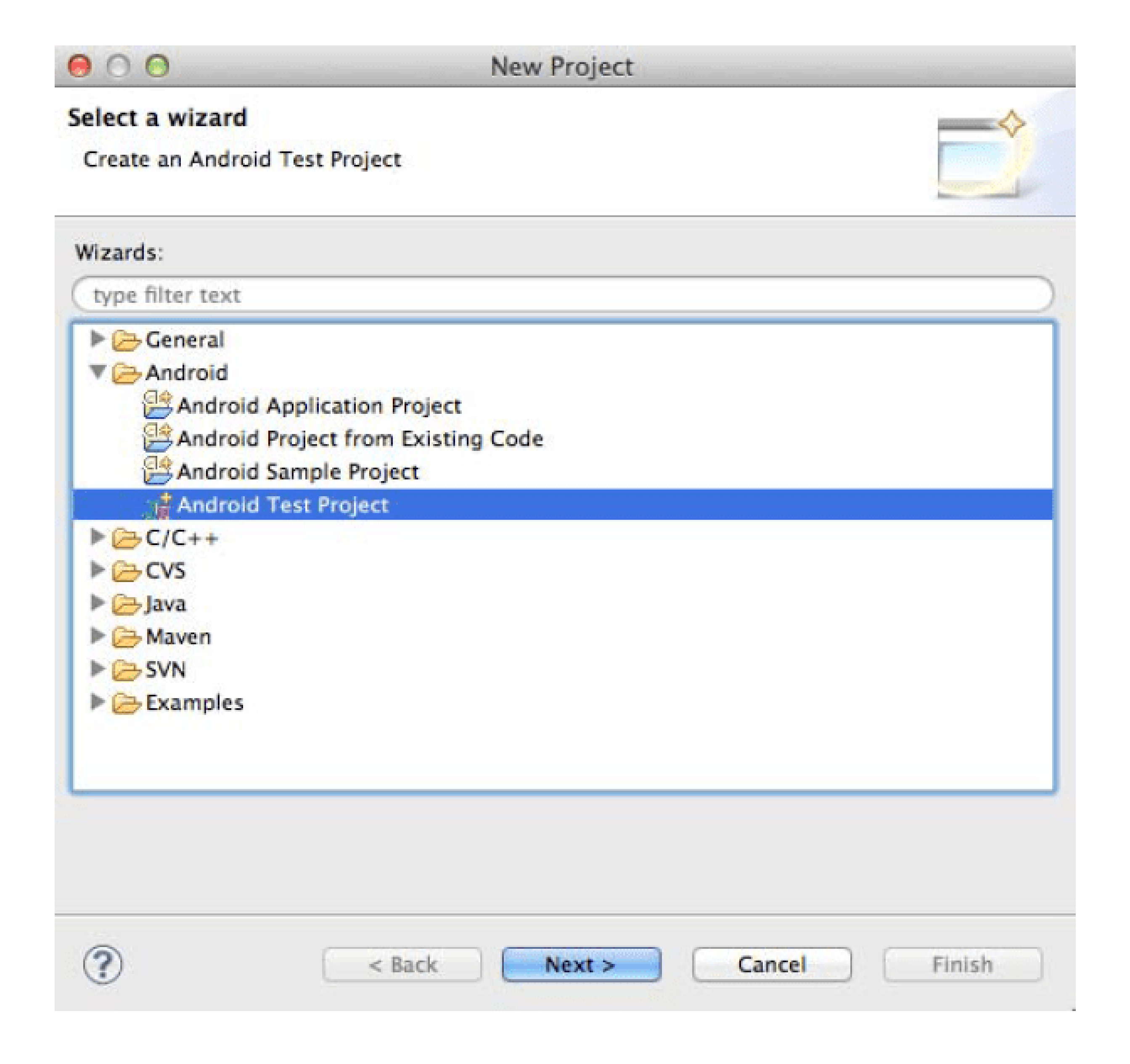
（3）接着通过 emulator 命令来启动模拟器。启动完毕之后，我们开始创建一个被测工程。在这个例子中，我们选择 Android 自带的项目工程 snake。在 eclipse 中选择 File→New→project···，随后选择 Android Sample Project，如下图所示。



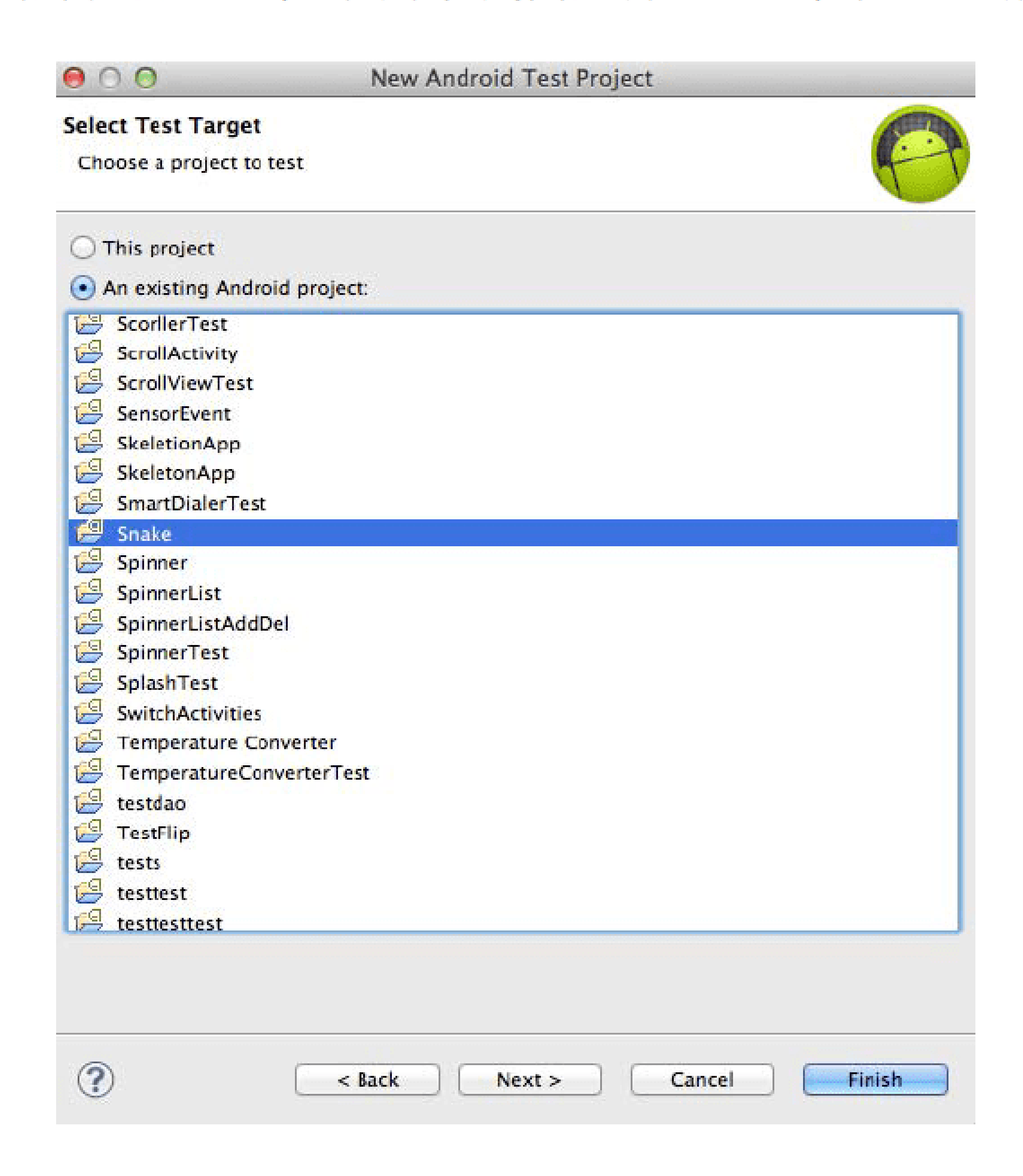
（4）单击“Next”按钮之后选择 Android4.1.2 和 Android 自带的 Snake 工程。



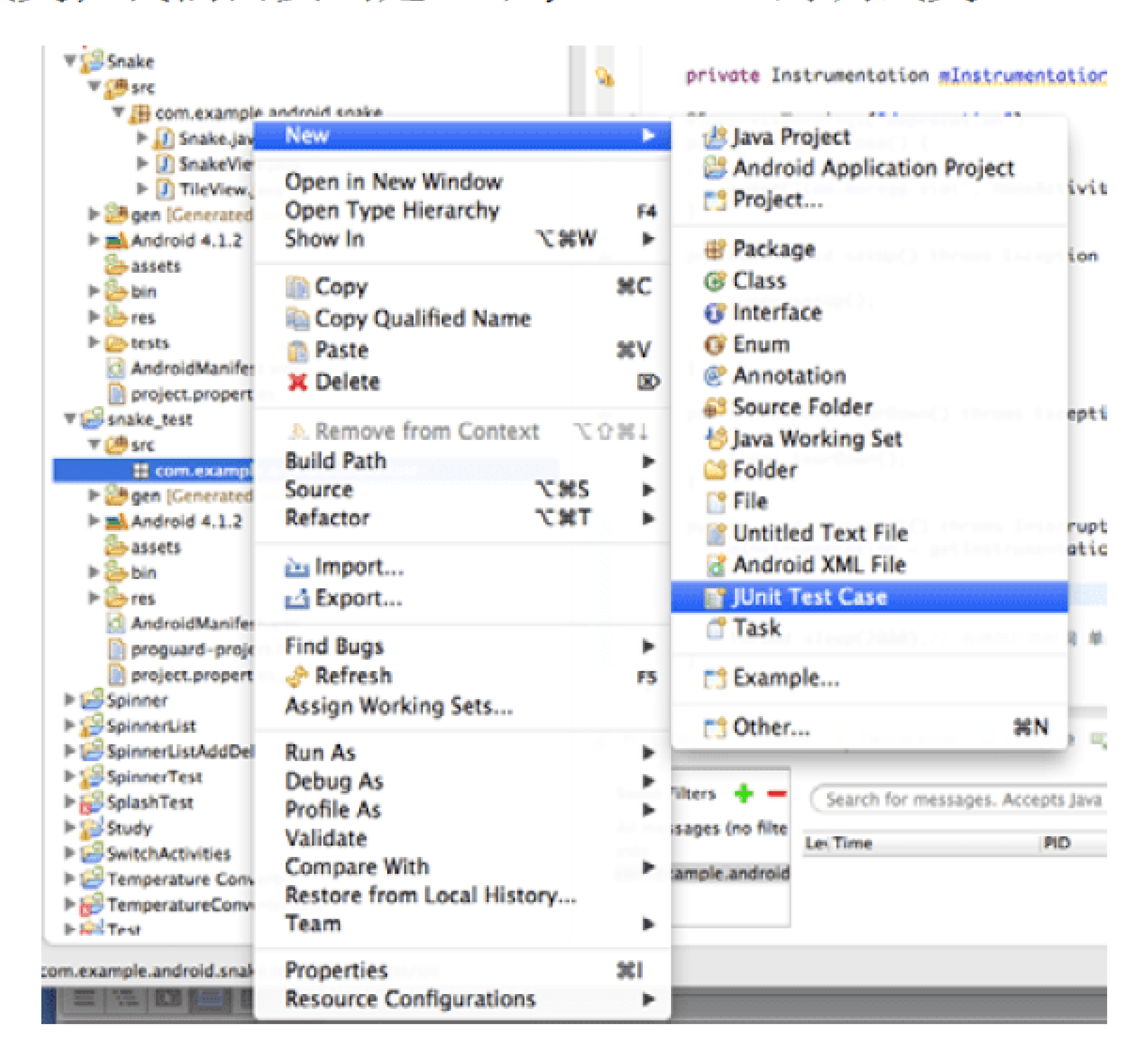
（5）创建好被测工程 Snake 之后，我们需要继续创建测试工程 Snake_test。单击 eclipse 中的 File→New→Project⋯ ，选择 Android Test Project，如下图所示。



（6）选择要测试的项目工程，本例中我们选择 Snake，如下图所示。

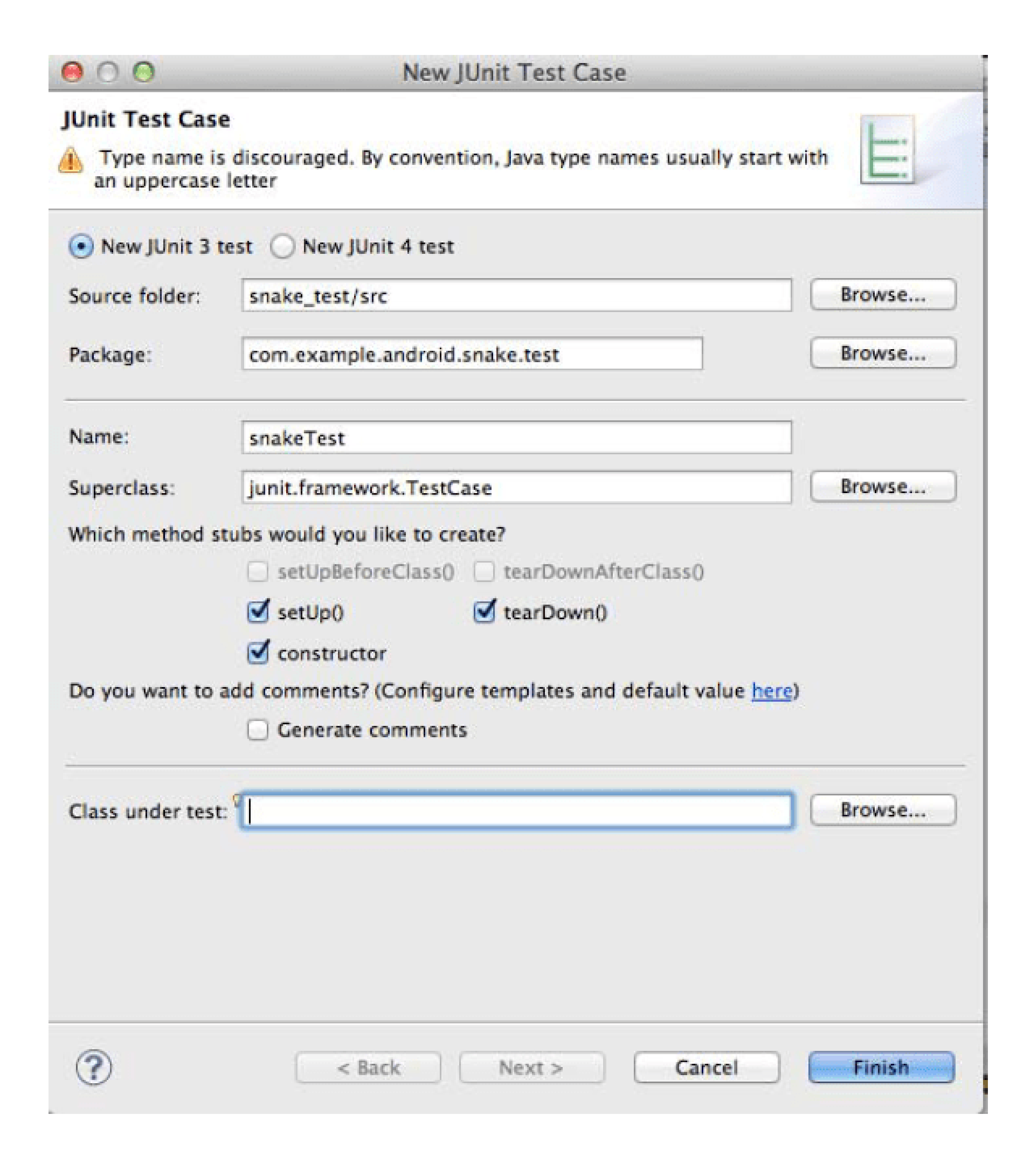


（7）这样我们被测工程 Snake 和测试工程 Snake_test 都已经创建成功。测试工程目前还没有任何测试类，我们需要创建一个 JUnit Test 的测试类 snakeTest，如下图所示。





类名为 snakeTest，如下图所示。



（8）创建完 snakeTest 类，根据 instrumentation android junit test 的规范我们添加如下代码：

```
package com.example.android.snake.test;


import android.annotation.SuppressLint;
import android.app.Instrumentation;
import android.test.ActivityInstrumentationTestCase2;


import com.example.android.snake.Snake;


import junit.framework.TestCase;
```

```
@SuppressLint("NewApi")
public class snakeTest extends ActivityInstrumentationTestCase2<Snake> {

        private Snake mActivity;

        private Instrumentation mInstrumentation;

        @SuppressWarnings("deprecation")
        public snakeTest() {

                super("com.example.android.snake", Snake.class);
        }

        protected void setUp() throws Exception {
                mInstrumentation = getInstrumentation();
                mActivity = this.getActivity();
                super.setUp();

        }

        protected void tearDown() throws Exception {

                super.tearDown();
        }

        public void testSnake() throws InterruptedException {
```

```
                    //这里添加测试代码
          }
}
```

(9) 一切准备工作已经完成了，接下来我们进入计算代码覆盖率 emma 工具的主要步骤。首先创建被测程序的 build.xml 文件，执行如下命令：

```
cd <main project folder>
android update project --path <main project folder>
```

(10) 成功执行之后在被测应用工程路径下会生成一个 build.xml 文件，并看到如下日志：

```
Updated local.properties
No project name specified, using Activity name 'Snake'.
If you wish to change it, edit the first line of build.xml.
Added file /Users/apple/Desktop/workspace/Snake/build.xml
Added file /Users/apple/Desktop/workspace/Snake/proguard-project.txt
It seems that there are sub-projects. If you want to update them
please use the --subprojects parameter.
```

(11) 同时需要生成测试程序的 build.xml 文件，执行如下命令：

```
android update test-project -m <full path to main project> -p <path to tes
t project>
```

(12) 成功执行之后在测试工程路径下会生成一个 build.xml 文件，并看到如下日志：

```
Resolved location of main project to: /Users/apple/Desktop/workspace/Snake
Updated project.properties
Updated local.properties
Added file /Users/apple/Desktop/workspace/snake_test/build.xml
Updated file /Users/apple/Desktop/workspace/snake_test/proguard-project.txt
```

```
Updated ant.properties
applematoMacBook-Pro:snake
```

（13）接着我们用如下命令安装两个应用的 apk 到 Android 模拟器上：

```
cd <path to main project>
ant emma debug install
cd <path to test project>
ant emma debug install
```

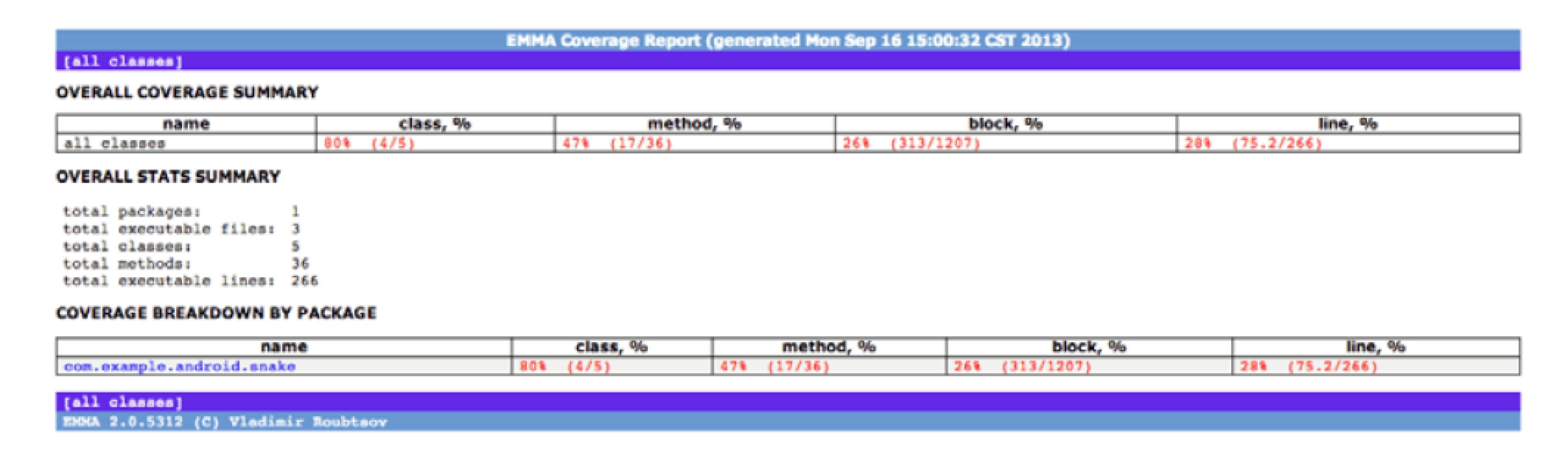
（14）最后使用 ant emma debug test，即可运行我们写好的 android junit test 代码，最终会生成代码覆盖率的 html 文件，可在日志中看到：

```
test:
[getlibpath] Library dependencies:
[getlibpath] No Libraries
    [echo] Running tests...
    [echo] Running tests ...
    [exec]
    [exec] com.example.android.snake.test.snakeTest:.
    [exec] Test results for InstrumentationTestRunner=.
    [exec] Time: 4.037
    [exec]
    [exec] OK (1 test)
    [exec]
    [exec]
    [exec] Generated code coverage data to /data/data/com.example.androi
d.snake/coverage.ec
    [echo] Setting permission to download the coverage file...
    [echo] Downloading coverage file into project directory...
    [exec] 95 KB/s (537 bytes in 0.005s)
```

```
    [echo] Extracting coverage report...
    [echo] Cleaning up temporary files...
  [delete] Deleting: /Users/apple/Desktop/workspace/snake_test/bin/cover
age.ec
  [delete] Deleting: /Users/apple/Desktop/workspace/Snake/bin/coverage.em
    [echo] Saving the coverage reports in /Users/apple/Desktop/workspace/
snake_test/bin

BUILD SUCCESSFUL
Total time: 13 seconds
```

从日志中能够看出，报告是由 coverage.ec 和 coverage.em 两个文件组合编译形成的，保存在测试工程的/bin 下面的 coverage.html 文件，不过由于本例在测试类中的 testSnake 方法中并没有写任何具体用例，所以功能几乎没有被覆盖，测试覆盖率报告如下图所示。



EMMA Coverage Report (generated Mon Sep 16 15:00:32 CST 2013)

[all classes]

OVERALL COVERAGE SUMMARY

name	class, %	method, %	block, %	line, %
all classes	80% (4/5)	47% (17/36)	26% (313/1207)	28% (75.2/266)

OVERALL STATS SUMMARY

```
total packages:          1
total executable files:  3
total classes:           5
total methods:           36
total executable lines:  266
```

COVERAGE BREAKDOWN BY PACKAGE

name	class, %	method, %	block, %	line, %
com.example.android.snake	80% (4/5)	47% (17/36)	26% (313/1207)	28% (75.2/266)

[all classes]

EMMA 2.0.5312 (C) Vladimir Roubtsov

（15）被测工程源代码的每个类都可以点击查看代码具体的覆盖率，如下图所示。

```
    /**
     * Updates the current mode of the application (RUNNING or PAUSED or the like)
     * as well as sets the visibility of textview for notification
     *
     * @param newMode
     */
    public void setMode(int newMode) {
        int oldMode = mMode;
        mMode = newMode;

        if (newMode == RUNNING & oldMode != RUNNING) {
            mStatusText.setVisibility(View.INVISIBLE);
            update();
            return;
        }

        Resources res = getContext().getResources();
        CharSequence str = "";
        if (newMode == PAUSE) {
            str = res.getText(R.string.mode_pause);
        }
        if (newMode == READY) {
            str = res.getText(R.string.mode_ready);
        }
        if (newMode == LOSE) {
            str = res.getString(R.string.mode_lose_prefix) + mScore
                  + res.getString(R.string.mode_lose_suffix);
        }

        mStatusText.setText(str);
        mStatusText.setVisibility(View.VISIBLE);
    }
```

覆盖率代码中的代码会以 3 种颜色来标识：红色标示的代码表示在测试执行中没有被运行到；黄色标示的代码表示在测试用例执行过程中部分被运行到，一般是方法中的判断语句等；绿色标示的代码则表示在测试执行中完全被运行到。整个报告一目了然，简单易懂。

在本例中有两个注意点：

- 如果在 Android 的 junit test 类中写了多个接口或是单元测试的话，那么最终的代码覆盖率就会根据具体的自动化用例计算。如果要实现黑盒功能测试来计算代码覆盖率的话，只要在测试方法中添加类似 thread.sleep()的方法即可。**在 sleep 的过程中，测试工程师可以操作被测应用，这样最终在用例执行完毕之后，同样能够生成手动用例所对应的代码覆盖率。不过需要注意的是**，如果被测应用在 junit 执行过程中崩溃的话，那默认的方式是不生成报告的。
- 如果在 Ant 编译过程中发现@override error 错误，需要关注 jdk 版本以及环境变量路径设置是否正确。

如果编译过程中出现 not found symbols，那是因为遗漏了被测程序所依赖的 lib 包，将 lib 包放入工程的/libs 即可。

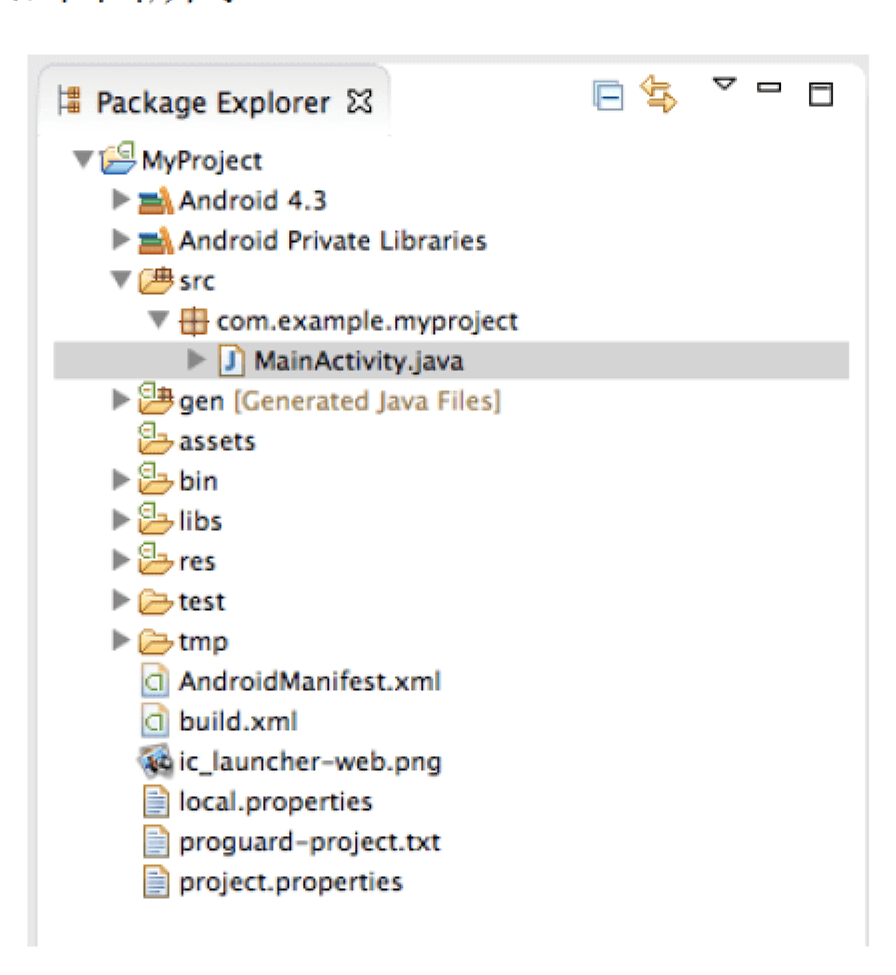
如果没有 emma.jar 的话，那么可以选择升级 SDK，或者前往下载：http://developer.android.com/sdk/index.html。

写到这里，本例就结束了。其实 emma 在整个 android junit test 中的使用非常简单，不过测试用例对于代码的覆盖率还是非常有价值的数据，不仅能让测试者知道用例哪里有欠缺，还能够让开发者进一步审视自己的代码。

6.3　robolectric

说到 robolectric，这是我非常推崇的一个框架，官方网站：robolectric.org。从框架本身的定义来看，其实 robolectric 是一个单元测试框架，但它的代码编写却与功能业务测试强绑定，其优势还不仅仅只有这些，接下来我们说一个很简单的无脑例子，从而揭开 robolectric 神秘的面纱。

（1）首先在 Eclipse 中新建一个被测试工程，工程名叫做 MyProject，类名是 MainActivity，工程目录如下图所示。



（2）在 MainActivity 中添加如下代码：

```
package com.example.myproject;

import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;

public class MainActivity extends Activity {
        public Button bt1;
        public TextView tv1;
        @Override
        protected void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                setContentView(R.layout.activity_main);
                bt1 = (Button) findViewById(R.id.button1);
                tv1 = (TextView) findViewById(R.id.textView1);
                bt1.setOnClickListener(new View.OnClickListener() {

                        @Override
                        public void onClick(View v) {
                                // TODO Auto-generated method stub
                                tv1.setText("the text is changed");
                        }
                });
        }
}
```

在该类中，我们将 Button、TextView 类的对象实例化之后，与 activity_main.xml 中的控件绑定定义一个点击的事件。当单击按钮之后，文本显示框内的文字就会改变。代码非常简单，这里就不多做解释了。

（3）接着我们来看之前提到的 activity_main.xml 文件，配置如下：

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android
"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity" >

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/hello_world" />

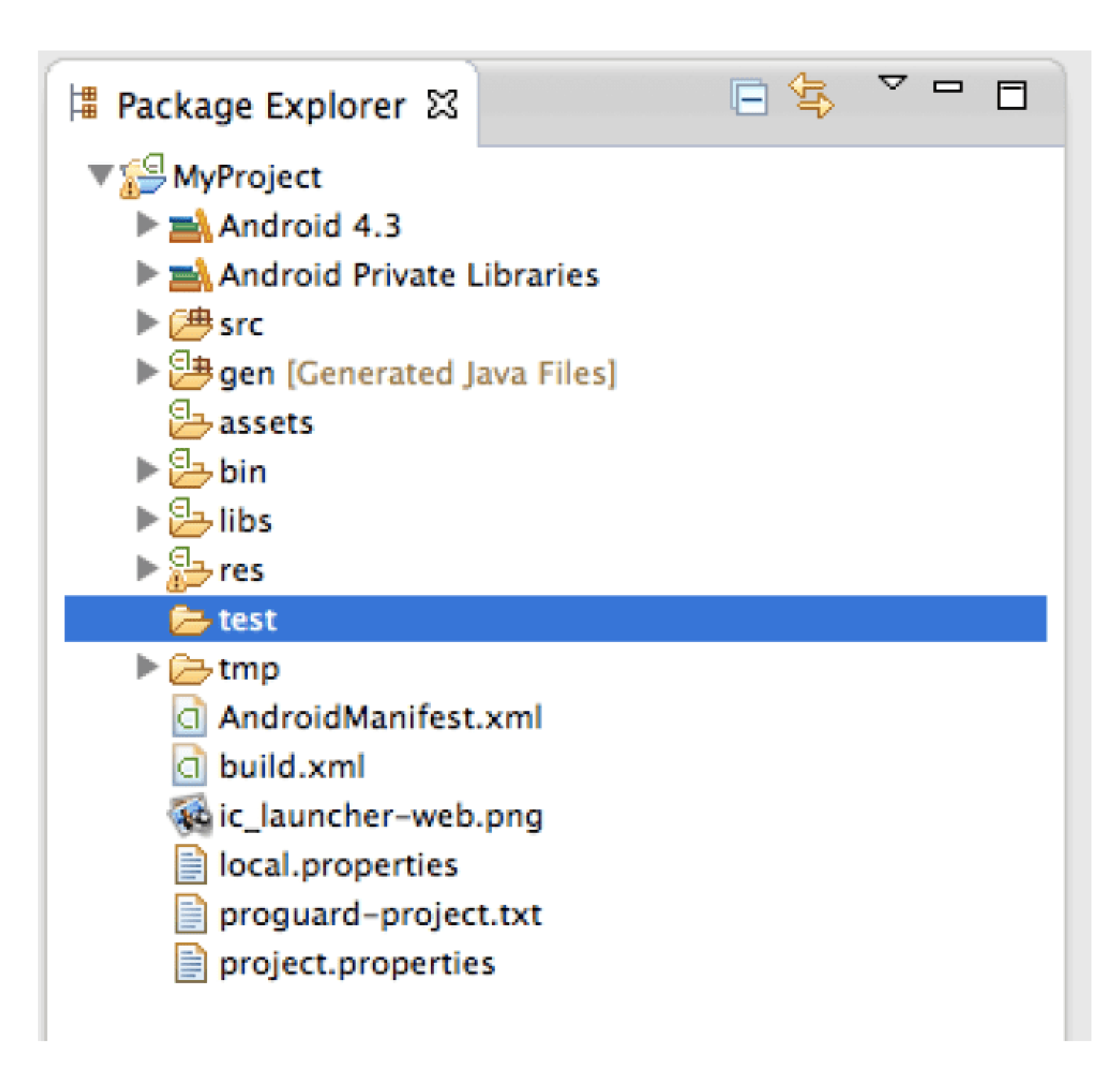
    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/textView1"
        android:layout_marginTop="72dp"
        android:layout_toRightOf="@+id/textView1"
```

```
        android:text="monkeytest" />

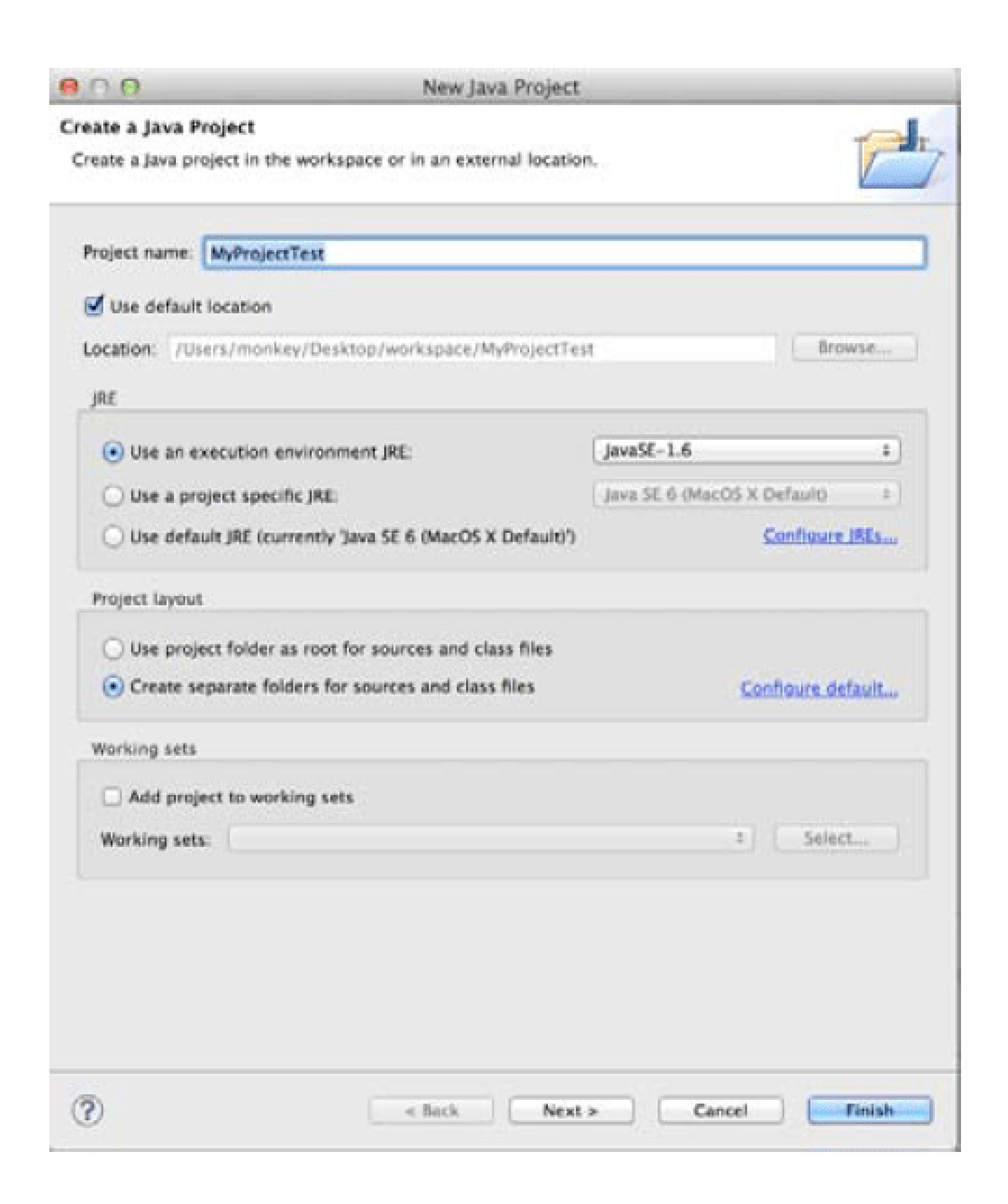
</RelativeLayout>
```

（4）到此为止，MyProject 这样一个被测试工程就算完成了

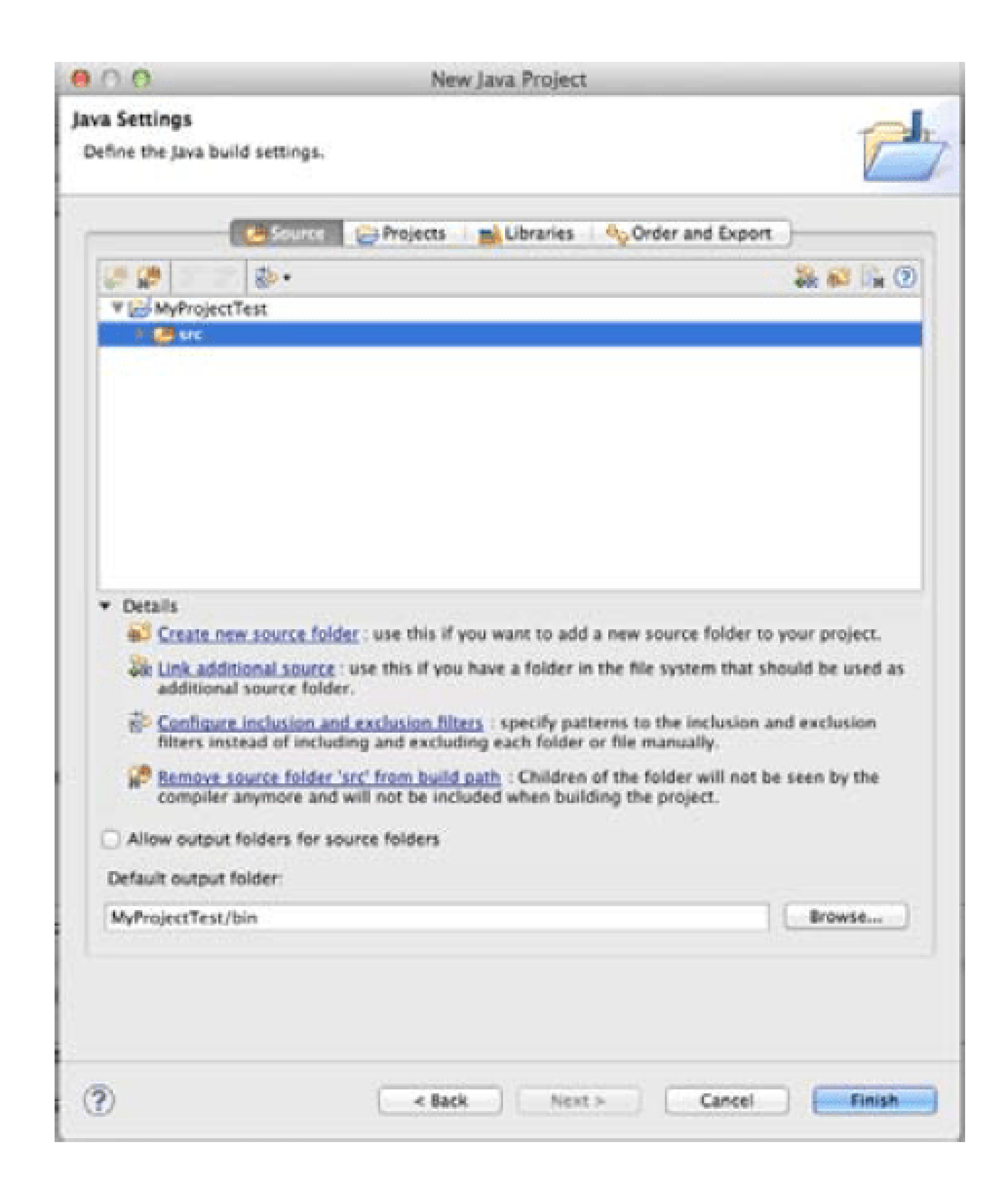
接下来就是一步一步地创建 roboletric 的测试工程。首先我们需要在被测工程中新建一个名叫 test 的文件夹，如下图所示。



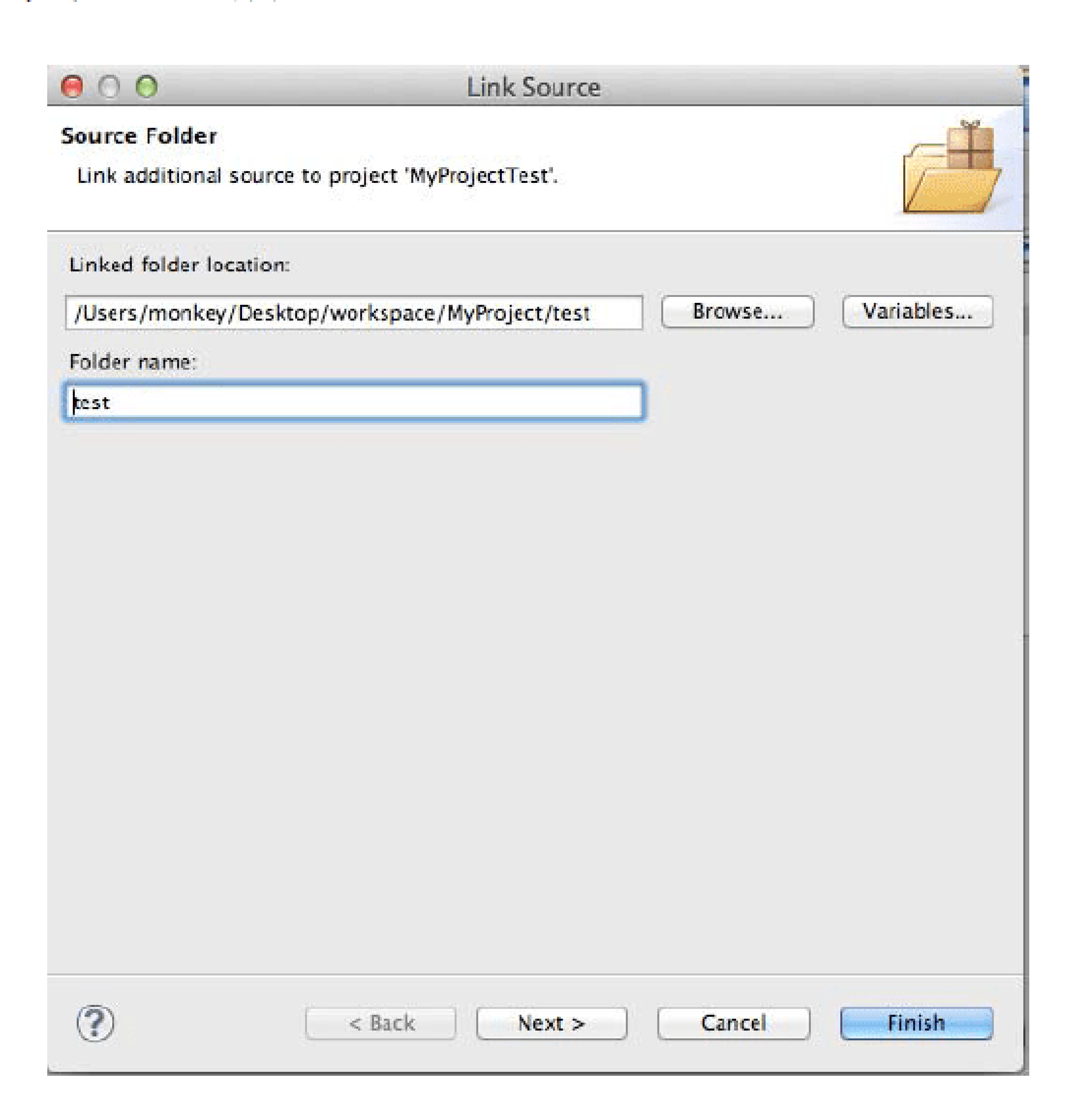
（5）接着新建一个 Java 工程，这里要强调一下是 Java 工程，我们取名叫做 MyProjectTest，如下图所示。



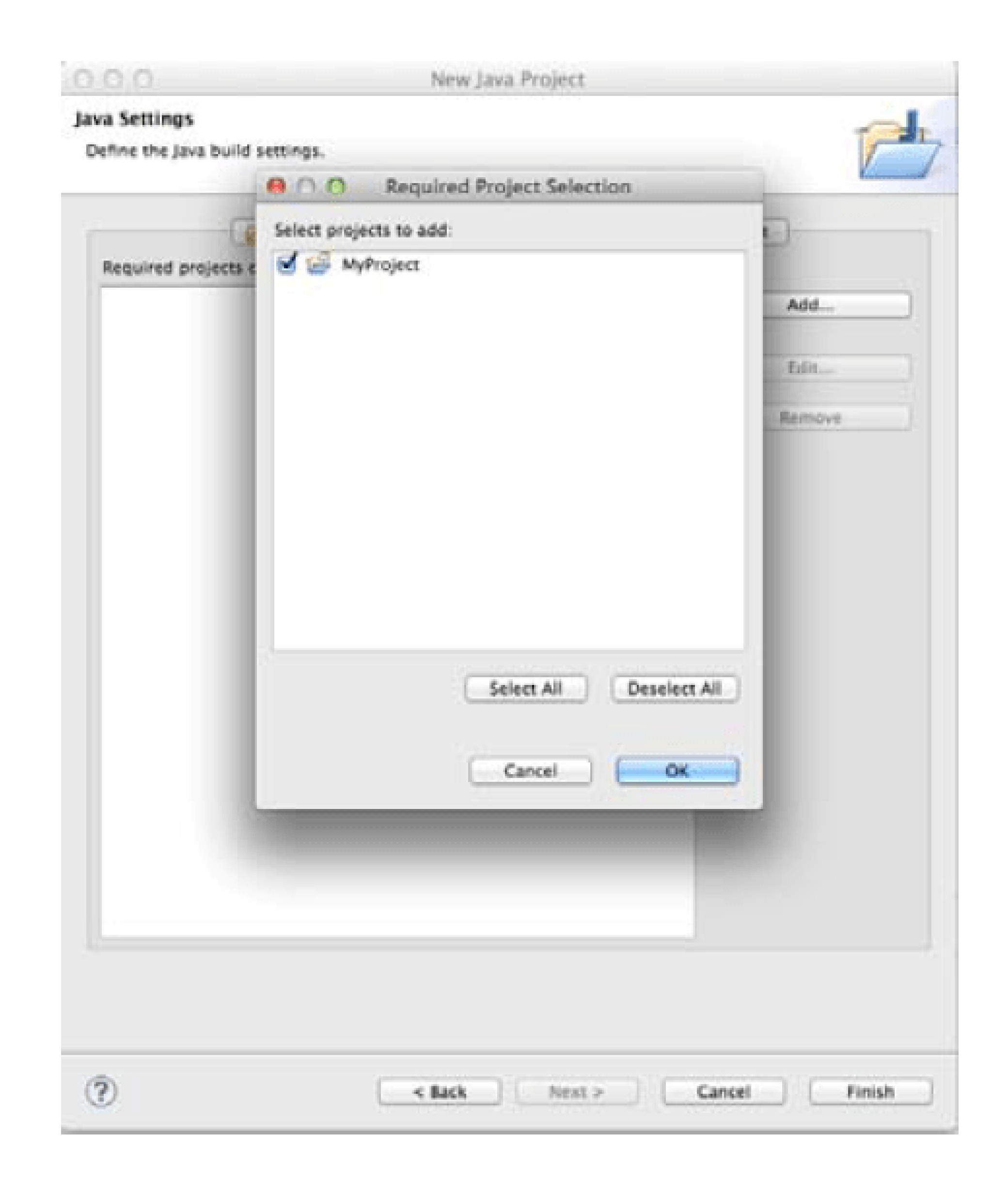
（6）单击“Next”按钮，选中 src，单击 Remove source folder 'src' from build path，如下图所示。



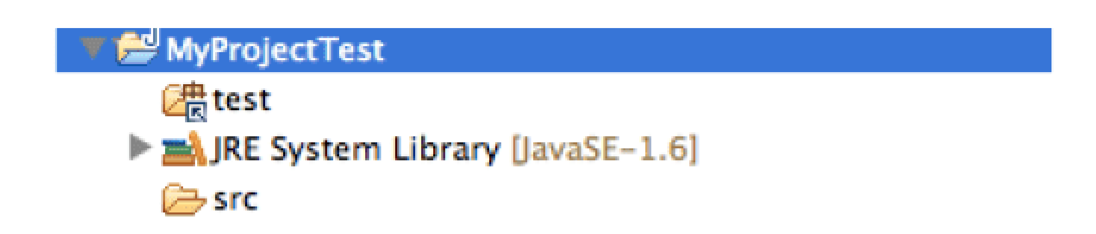
（7）单击 Link additional source 并将路径选择为被测工程 MyProject 里的 test 文件夹，单击 Finish 按钮，如下图所示。



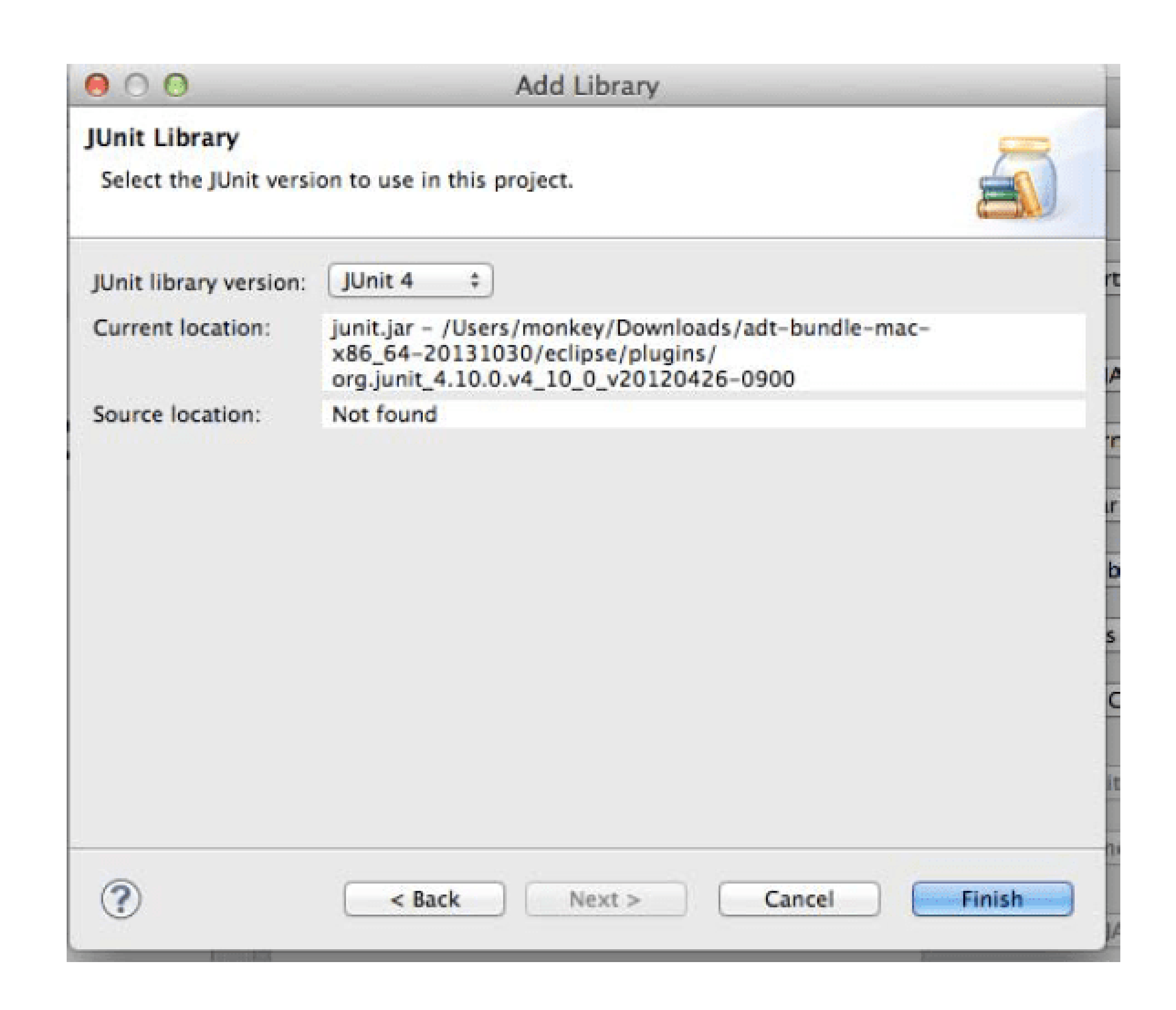
（8）选择最上方的 Projects 标签，单击 Add 按钮，选择被测工程 MyProject，单击 OK 按钮，单击 Finish 按钮。如下图所示。



（9）如果一切设置成功，就会看到成功地创建了一个 Java 工程，如下图所示。



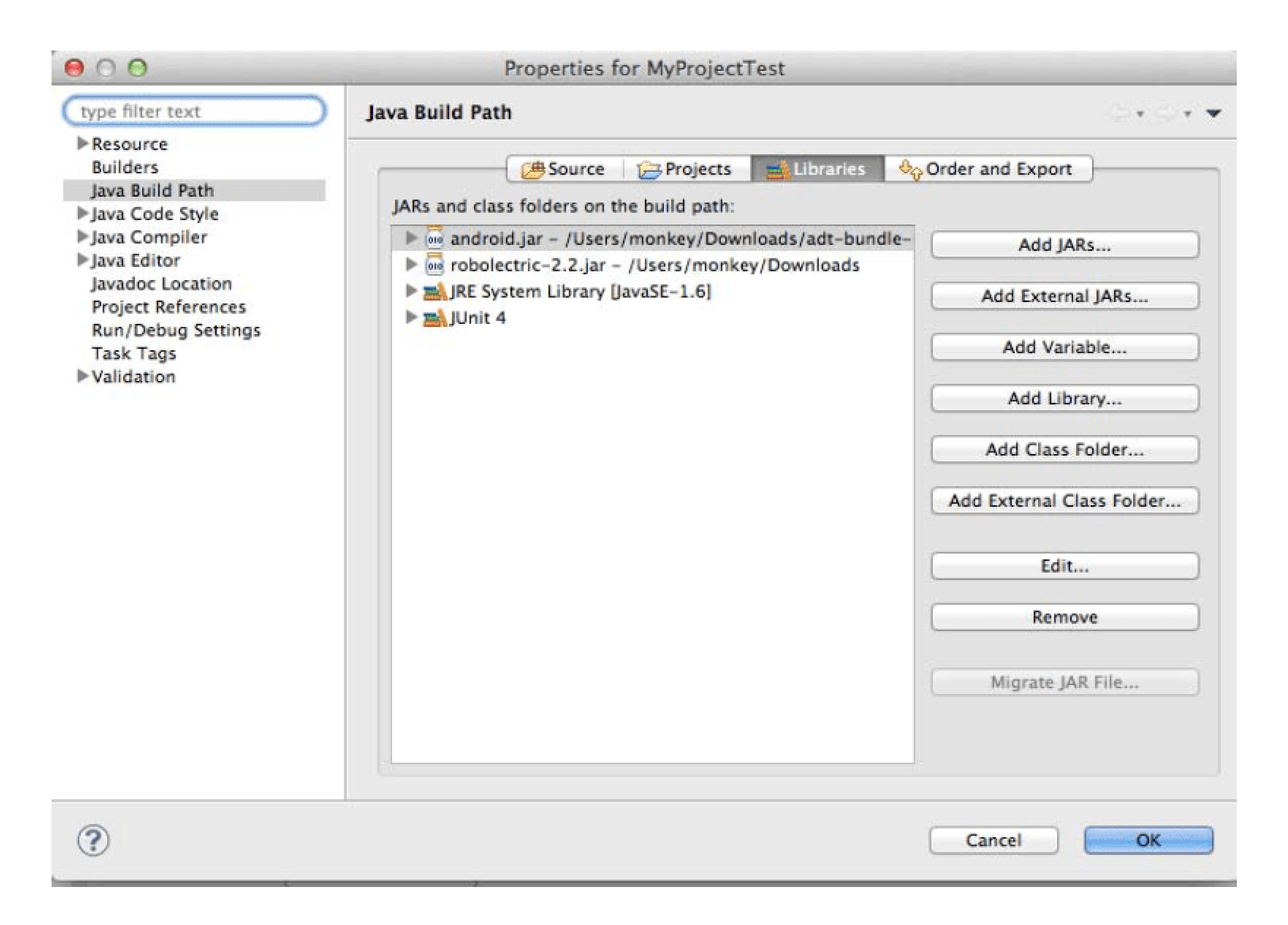
（10）接着我们再来添加一些 MyProjectTest 工程相关的依赖包。打开 Configure build path，单击 Add Library 按钮，添加 Junit4 的依赖包，这里需要强调一下 Robolectric 并不兼容 Junit3。



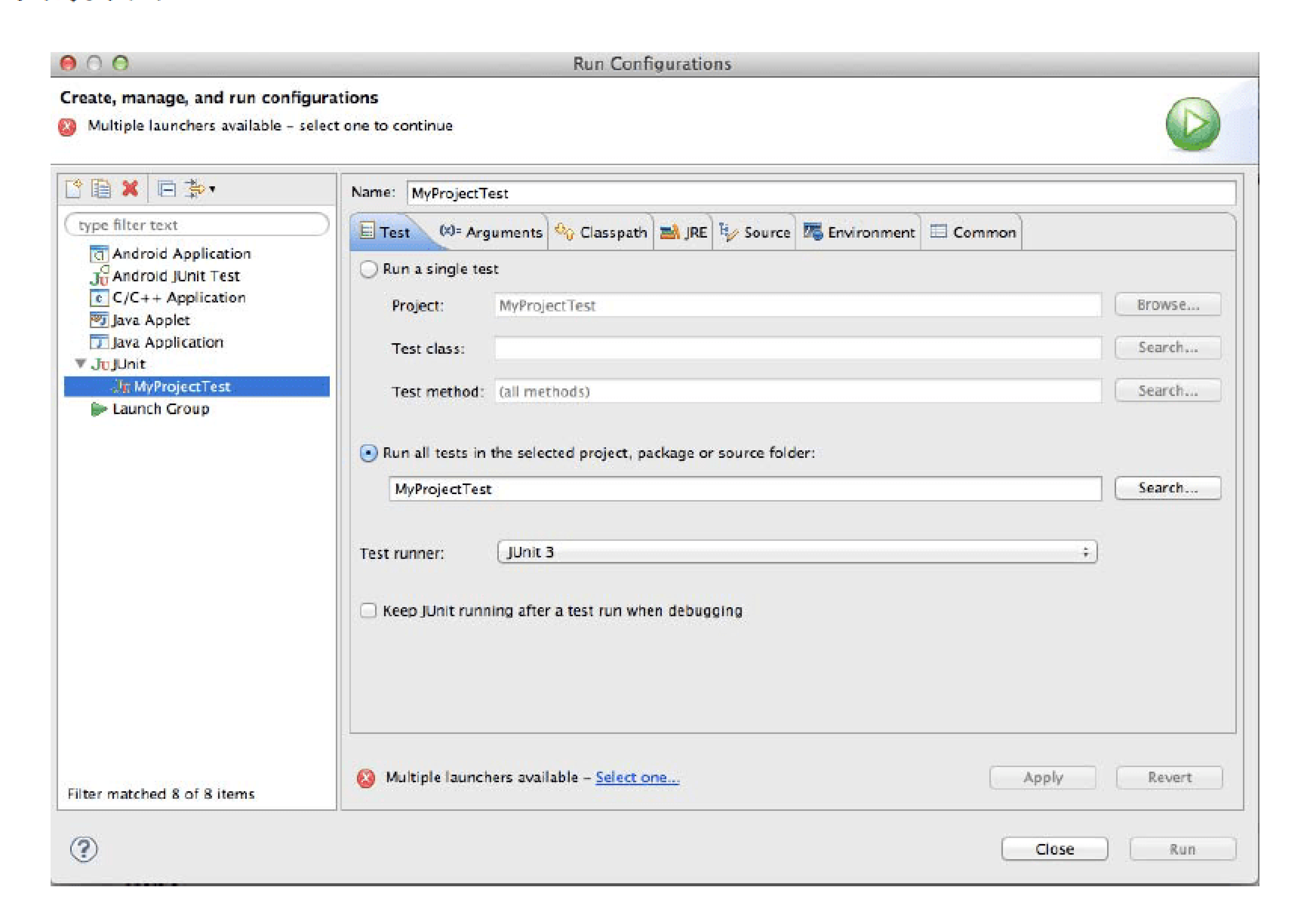
（11）接着添加 Robolectric 的主要依赖包，下载地址：

http://search.maven.org/#search%7Cga%7C1%7Cg%3A%22org.robolectric%22

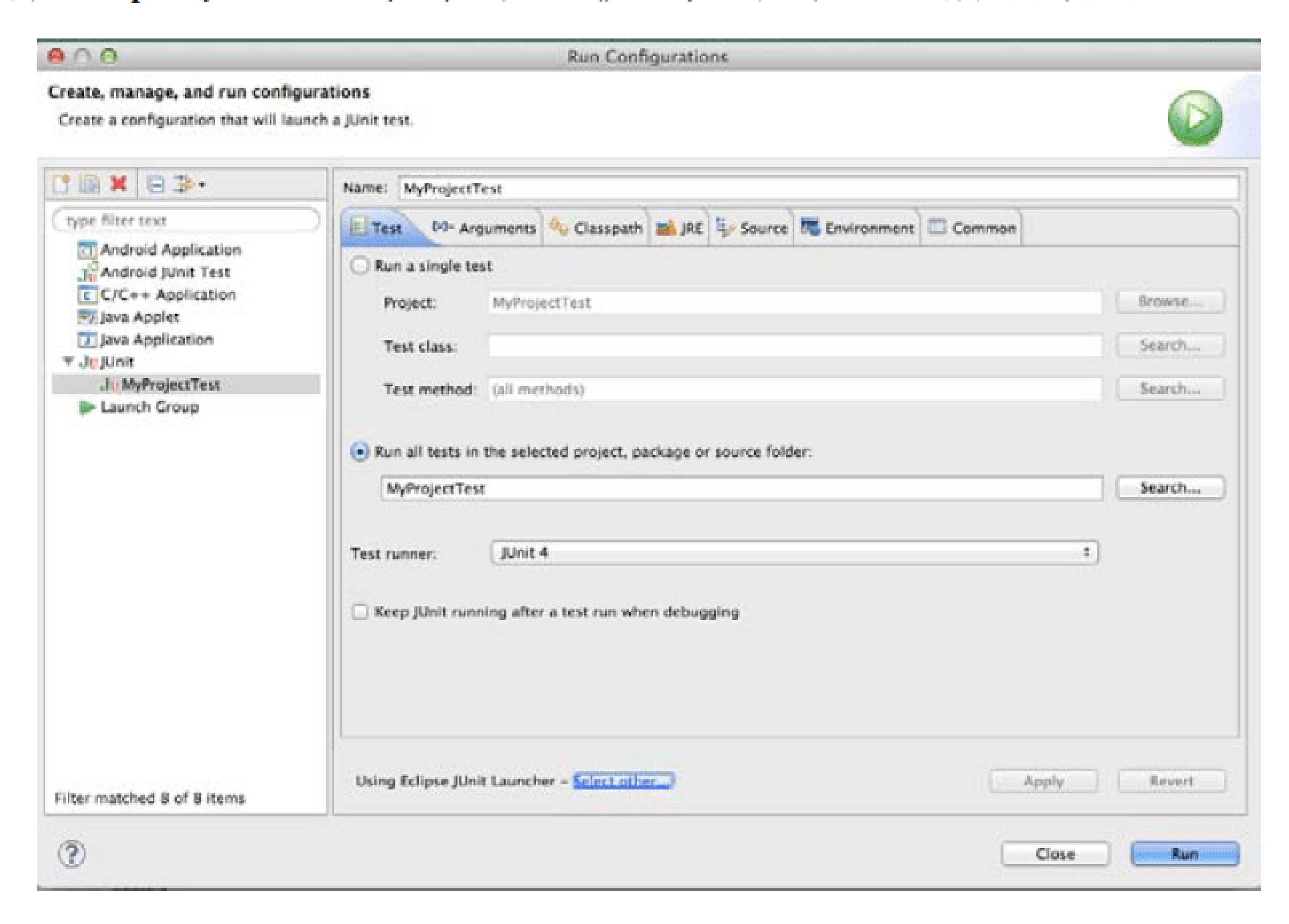
（12）还需要添加一个与被测工程 Android 版本对应的 Android 的依赖包 Android.jar，这个包能够在 sdk 的/platforms 目录下找到，单击 Finish 按钮后，出现如下图所示的界面。



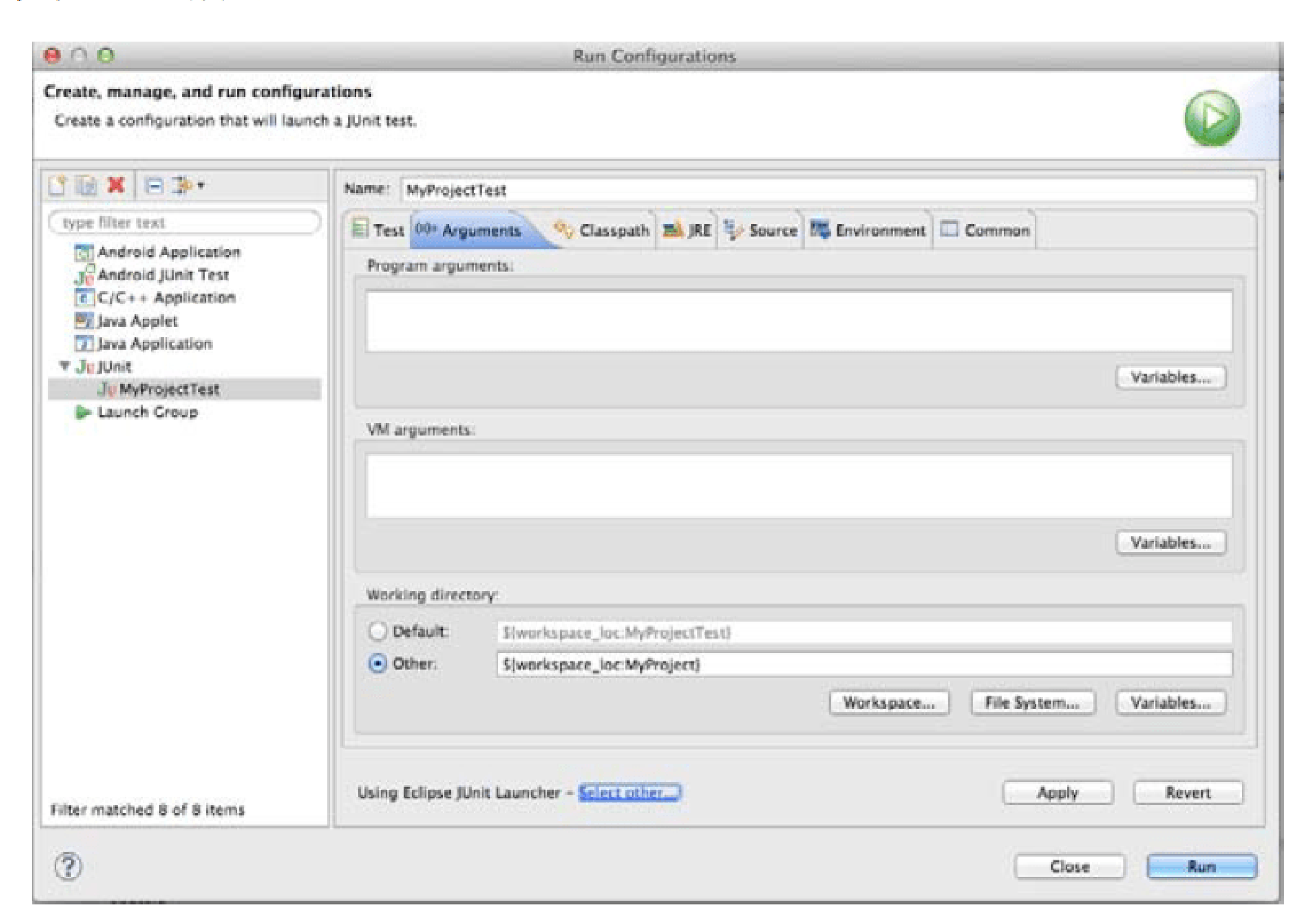
（13）至此，测试工程的依赖包已经全部设置完成，最后我们还需要创建一个 test Run Configuration，右键测试工程 Run→Run Configurations...，双击 Junit Test，可看到下图所示的界面。



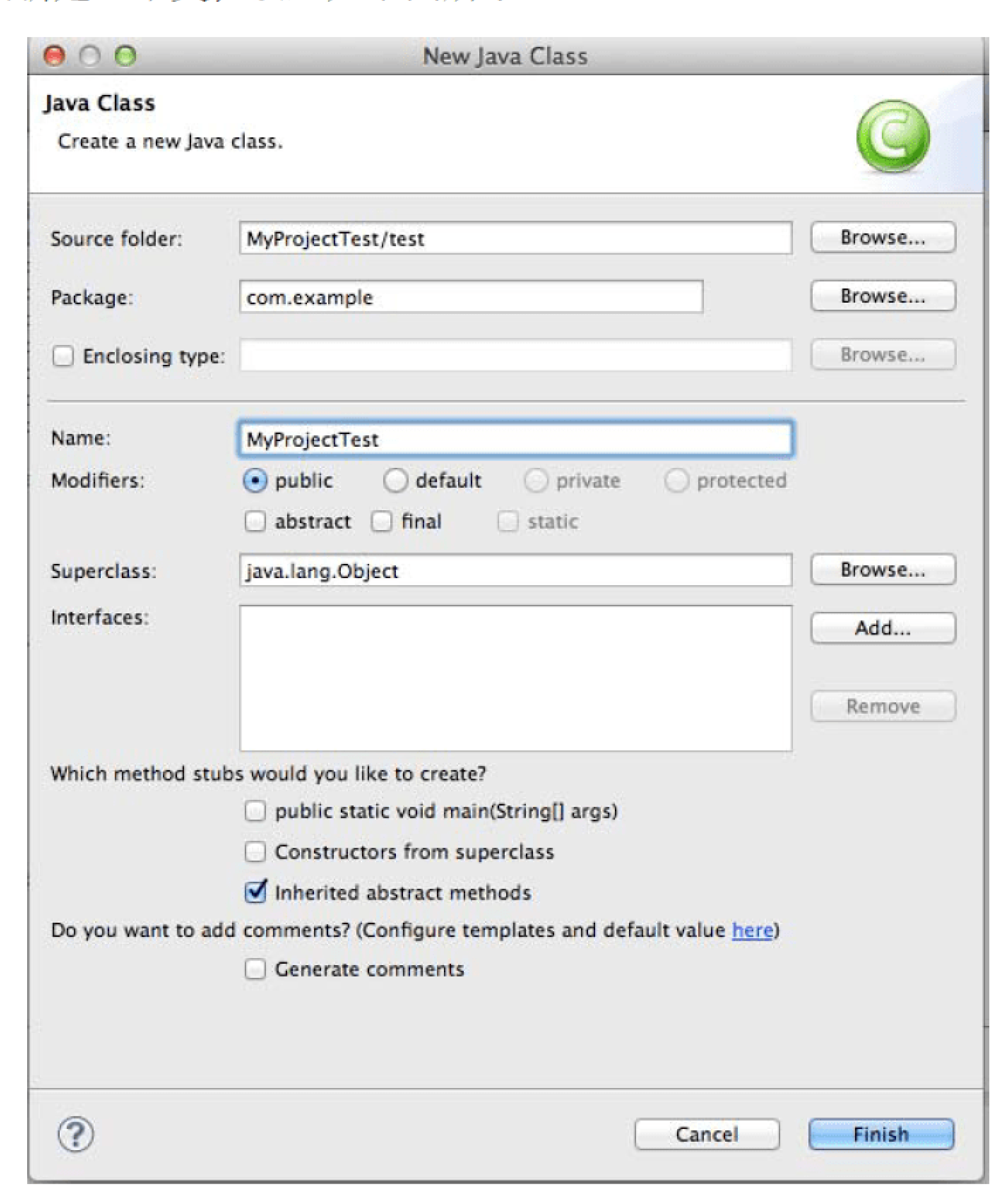
（14）更改 Test runner 为 Junit4（毋庸置疑），选择 Multiple launchers available Select one...，选择 Eclipse Junit Test，单击 OK 按钮，出现如下图所示的界面。



（15）选择 Arguments 标签，在 Working directory：中选择 other 并选中被测工程 MyProject，如下图所示。



（16）单击 Close 按钮之后，MyProjectTest 工程的所有设置工作结束，最后我们需要来编写测试代码（是不是觉得过程很麻烦，没有关系，熟练就好啦）。右键测试工程的 test 文件夹新建一个类，设置如下图所示。



（17）单击 Finish 按钮之后，添加如下代码：

```
package com.example;


import android.widget.Button;
import android.widget.ImageView;
import android.widget.TextView;
```

```
import com.example.myproject.MainActivity;
import com.example.myproject.R;

import org.robolectric.Robolectric;
import org.robolectric.RobolectricTestRunner;
import org.junit.BeforeClass;
import org.junit.Test;
import org.junit.runner.RunWith;

import static org.hamcrest.CoreMatchers.equalTo;
import static org.junit.Assert.assertThat;

@RunWith(RobolectricTestRunner.class)
public class MyProjectTest {

private MainActivity activity;
private Button testButton;
private ImageView pivotalLogo;
private TextView testText;



@Test
public void shouldHaveActivity() throws Exception {
      activity = Robolectric.buildActivity(MainActivity.class).create().g
et();
      assertThat((Class<MainActivity>) activity.getClass(), equalTo(MainA
ctivity.class));
```

```
}

    @Test
    public void shouldHaveHappySmiles() throws Exception {
        String hello = new MainActivity().getResources().getString(R.strin
g.hello_world);
        assertThat(hello, equalTo("Hello world!"));
    }
    @Test
    public void shouldHaveAButtonThatSaysPressMe() throws Exception {
           activity = Robolectric.buildActivity(MainActivity.class).create
().get();
            testButton = (Button) activity.findViewById(R.id.button1);
        assertThat((String) testButton.getText(), equalTo("monkeytest"));
    }
    @Test
    public void shouldHaveClickButtonSuccess() throws Exception {
       activity = Robolectric.buildActivity(MainActivity.class).create().g
et();
        testButton = (Button) activity.findViewById(R.id.button1);
        testText = (TextView) activity.findViewById(R.id.textView1);
        testButton.performClick();
        assertThat((String) testText.getText(),equalTo("the text is change
d"));


    }
```

```
}
```

大部分同学肯定已经看出来了，这代码根本就是第 2 个 robotium，只不过感觉在代码编写的规则上不同而已，这也正是 robolectric 的精妙之处：看似业务测试的代码却可以像单元测试一样执行和验证。Robolectric 运行的是 Junit Test 并非 Android Junit Test，所以不需要去运行模拟器或者依赖真机。其源码中的 shadowActivity、ShadowXXX 的概念很好，可以说是从另外一个角度去剖析了 Android，将界面上的操作全部做了一层 Mock。



提示： 更多信息可以查看：
http://robolectric.org/eclipse-quick-start.html。

6.4　小　结

现在的框架实在太多，本章就不一一做列举了。大部分框架的核心以及思路都是相通的。常用的框架我再推荐几个：Epresso、cafe、cucumber、Appium、Athrun、NatvieDriver 等，更多信息可关注各个工具官方网站，有问题可直接私信我或在 www.testerhome.com 上留言，我们会第一时间回答您的问题。

读书笔记

第 7 章　移动应用测试案例实践分析

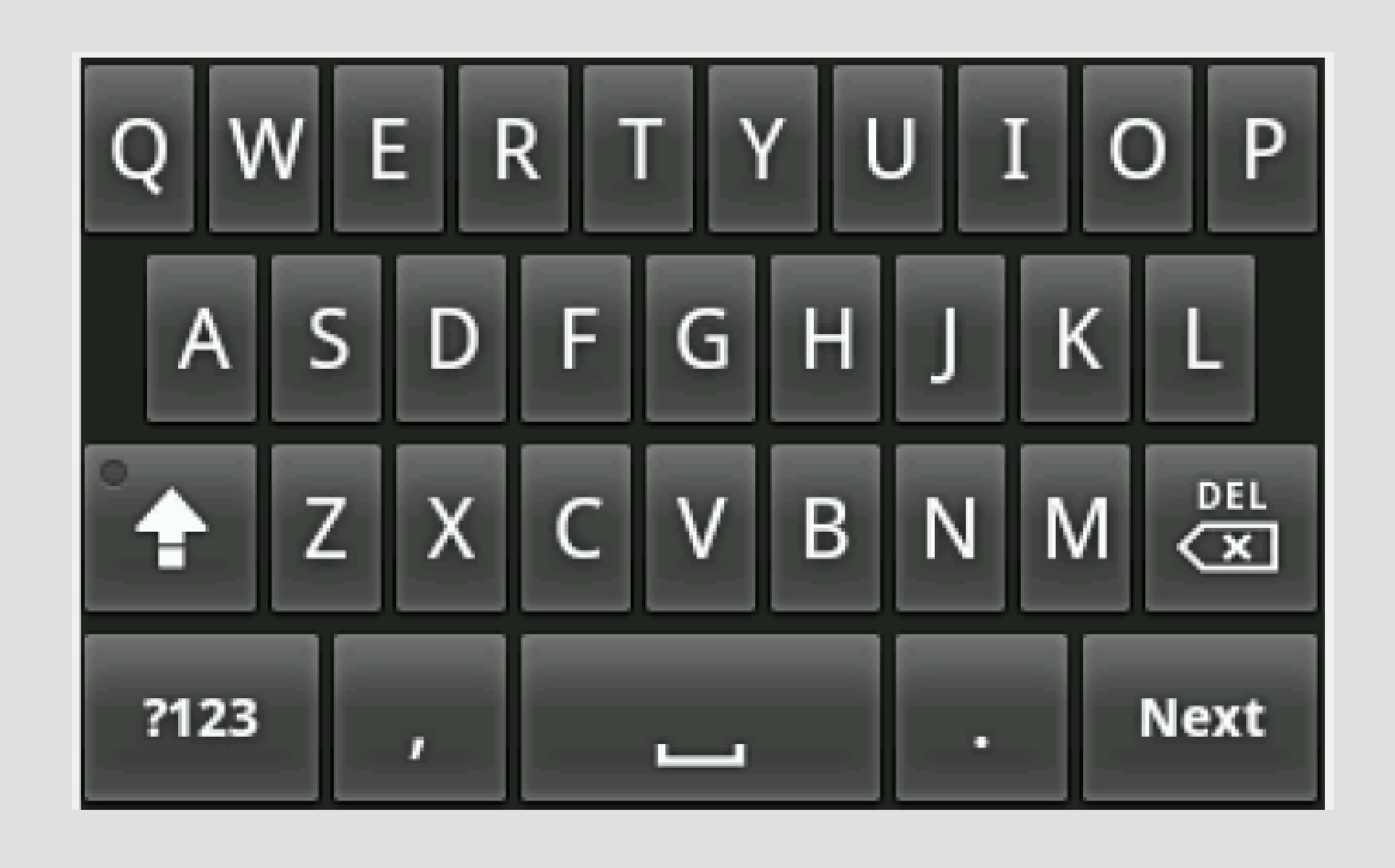
传统软件和传统互联网测试都十分注重测试用例。在面试软件测试工程师的时候，一般会着重考察撰写测试用例的功底。同样，在移动互联网应用测试行业，测试用例也是必不可少的。

那么在移动互联网应用的测试用例设计方法有哪些呢？需要注重哪些呢？传统测试用例在设计时候，一般只要抓住需求和覆盖分支就可以了，而移动应用测试用例的设计除了这两点外，更应该注重用户场景和测试环境。本章会举出一些实际的测试项目和场景与朋友们一起进行测试用例的设计，从而给出一些设计用例的思考方法。

7.1 深入了解被测试对象

很多从事移动互联网测试工程师的都会问："测试用例到底该如何设计？为什么我设计的用例总有遗漏？"笔者也一直犯错误，用例也一直有遗漏。必须说明的是，测试工程师写测试用例的目的是让产品尽可能地接近没有缺陷，接近用户心目中期望的完美产品，但也只能是接近，而不是成为，所以，有遗漏是很正常的现象。我们要求尽量避免更多的遗漏，这就要求我们更多地去了解与被测对象相关的一切。否则谈何测试？

故事场景：假设我们现在需要测试一个移动端的输入法应用，需要支持 10 个国家的语言。那么，作为一个测试工程师应该如何进行测试用例的设计呢？如何展开整个测试活动呢？



现在大多测试工程师们关心的问题有如下一些：

- 自动化测试怎么做？
- 性能测试怎么做？
- 除了功能测试，还能做什么？
- 要用什么工具吗？

以上这些的确都是问题，但最关键的问题却没有提出，那就是真正了解这个产品的一些基础的知识：

（1）产品有什么特点?

（2）产品所在的系统的结构?

（3）产品本身的系统架构?

（4）用户最关心产品的功能是什么?

现在让我们一个一个来解答看看。

第 1 个问题是：产品有什么特点？

回答：产品是移动端上的输入法。输入法最基本的特点莫过于每个国家的键盘布局、支持字母都不同。虽然产品在界面显示、其他附加功能上可以做个性定制，但是语言本身的特点是不会改变的。

第 2 个问题是：产品所在的系统结构？

回答：我们要测试的产品所处的系统各式各样，有 Android、iOS、Windowsphone、Blackberry 等。用户使用产品不会脱离所在的系统，那么我们就非常有必要去了解这些系统。比如，输入法安装对用户来说和一般应用没有什么区别，而从测试的角度去看就要清楚地了解输入法的各个资源文件、.so 文件保存在系统哪个文件夹中、结构的权限分别是什么等等。

第 3 个问题是：产品本身的系统架构？

回答：输入法本身是一个应用，但在 Android 系统中，输入法不是一个 Activity，而是一个 Service 。在这个 Service 中也许也会包含着若干个 Activity，比如设置界面等。对于测试工程师而言，需要清楚地知道这些内容。

第 4 个问题是：用户最关心产品的功能是什么？

回答：这也是移动互联网测试最应该关注的问题。就输入法而言，用户最关心的问题自然是取词是否又快又准、是否支持词库升级和词库备份、是否不必改变自己原本的

输入习惯等等。这些看上去像是非常普通的需求，但就现状而言，能够很好地满足广大普通用户的输入法应用真是不多。

让我们回到最开始的问题，自动化测试、性能测试、压力测试、持续集成固然很重要，但再怎么重要，测试工程师也要优先了解清楚上述的 4 个问题。这样一来摆在我们面前的问题就很明确了，无论你有多少年的工作经验，当面对自己不熟悉的测试对象的时候，都需要先要问问自己了解多少。因此，在进行输入法应用测试之前，根据调查我们需要清楚以下几点：

- 每种语言的输入法常用的布局有哪些？用户最常用的是哪些？
- 每种语言的输入法包含的基本字母有哪些？分别有什么特征？比如希腊语就有两个字母的大写一模一样。
- 每种语言的输入法用户输入习惯有什么不同。比如阿拉伯语输入显示就是从右往左显示等。



提示：也许很多朋友会说这些信息肯定会在需求说明书上写清楚的。传统软件行业我不清楚，但进军移动互联网的企业基本上都还处于各种摸索阶段。退一步说，就算真有非常详细的需求说明书，作为测试工程师也需要去了解这些内容，去验证这些需求是正确的。笔者一直贯穿一个宗旨，眼见为实。

7.2 多种数据来源

在平时，我们使用移动应用的时候，接触最多的就是数据，比如最基本的联系人、短信、照片、视频等。对这些看似平常的数据在测试的时候却要特别注意。由于这些数据有不同的来源，可能会出现很多我们意想不到的测试用例。很多产品的测试过程中都会和这些数据打交道，就拿图片举例（因为笔者现在做的是和图片有关的产品比较熟悉。）现在很多产品应用都有拍照、载入图片、裁剪图片、上传图片等功能，首先我们

需要为设计测试用例做好本章 7.1 节所说的一切（7.1 节所说的内容是一切测试活动正式开始的基础，在本书后面的内容就不再强调这一点了），然后我们开始设计测试用例。

故事场景：移动互联网某应用中有一个模块是和图片相关的。模块的功能是允许用户在签到、发信息、回复等操作时都能够带图片（这类功能在现在的应用中是非常常见的）。那该如何进行测试用例的设计呢？

作为测试工程师，我们会如何开始设计测试用例呢？拍脑袋先想到的有以下两点：

- 用移动设备直接拍照。
- 用移动设备载入相册中的图片。

这两点也是用户最常用、最常见的场景。但如果只有这两个场景的话，一切就会变得简单许多，也就不会出现用户可以重现，而测试工程师这里不能重现的情况了。所以继续思考之后，就会又想到以下的因素：

- 不同分辨率的图片。
- 不同格式的图片。
- 各种大小的图片。

至此，可以说已经在照片的大小、格式上运用了一些测试设计的思维方式了，但在移动互联网测试中依然还是远远不够的。随即我们继续思考，看看移动互联网用户更多的使用习惯是什么，从而得出了以下结论：

- 很多美女喜欢自拍，那么需要考虑会有前置摄像头的情况。
- 用户喜欢拼图。
- 用户大都喜欢使用自己美化过的照片，比如用美图秀秀、carmera360 等软件修图。
- 随着 iPhone 越来越多地被使用，全景图片也会越来越多。
- 随着智能机的普及，更多的用户喜欢把电脑中的图片导入到手机上。

- 用户会从其他各种应用或者网站直接将图片保存到本地。

随着我们不停地思考、推敲之后，似乎用例已经考虑得比较完善了。但用户智能机中的图片来源仅仅只有这些吗？是否还有可能有别的用户感觉不到、却经常发生的场景呢？经过调查，我们进一步得到以下的结论：

- 智能机中的图片会来自于某些应用的临时文件。
- Google、FaceBook 等邮箱、应用的绑定，会在用户登陆智能机的同时，将用户帐户下的图片同步到本地。
- 某些网页会将临时文件保存在本地。

最后，作为测试工程师必须要研究一下这些经由不同途径得到的图片，除了我们肉眼看得到的长宽比、像素不同之外，还有哪些不同。比如的这些数据会对应用产生什么影响，如果某个值没有的话会怎么样。

Image Metadata

TIFF Metadata:

```
           ResolutionUnit: 2
PhotometricInterpretat... 2
                 Software: Adobe Photoshop...
                 DateTime: 2013:03:05 00:03:01
              XResolution: 72
              Orientation: 1
                    Model: HTC Desire
                     Make: HTC
              YResolution: 72
```

Exif Metadata:

```
        DateTimeDigitized: 2013:03:04 20:26:27
          PixelXDimension: 828
         DateTimeOriginal: 2013:03:04 20:26:27
          PixelYDimension: 1076
               ColorSpace: 1
              FocalLength: 4.31
```

IPTC Metadata:

```
              DateCreated: 20130304
              TimeCreated: 202627
```

在移动互联网的项目中，我们会接触到各式各样的数据，而这些数据的来源随着应用的剧增和平台的多样性会变的越来越多。图片的例子仅仅是其中一个很小的例子，并且上面列举的还不是全部的测试用例项。笔者想告诉大家的是，我们需要用这样一种思维方式设计测试用例，否则在移动互联网的应用测试中我们会遗漏很多测试点。

7.3　在生活中使用产品

随着移动互联网的发展，现在越来越多的企业没有移动无线测试工程师这样一个职务，也有越来越多的相关从业人员涌入其中。这些测试人员可能每天都在测试一款应用，可能每天都在为这款应用汇报缺陷，也可能每天都抱怨这款应用的用户交互体验。但是一旦他们下班，很可能还没踏出公司的大门，就会把这款应用卸载了。他们了解自己测试的应用，但是他们不爱它，甚至讨厌它。如果你不是移动互联网的从业人员，那么请不要惊讶，这的确是事实。

从功能和占用空间上讲，移动互联网的产品通常不会是巨无霸，而是小而美，其目标群体是广大普通移动互联网用户，并且随时准备融入他们的生活。作为测试工程师，更需要将测试的产品融入到自己的生活中去，哪怕之前我们的生活和这个应用没有交集。当你开始尝试这样做了之后，你会发现另外一片天地。

故事场景：移动互联网中，智能机一个很常用的功能就是打电话。由于现在社会信息泄漏严重，所以某应用推出了一个“黑名单”功能。该功能可以阻止加入其列表中的号码或联系人再次与智能机主人通话。那我们要如何对这个功能做测试用例设计呢？

为什么这里要重点说说需要在生活中去使用我们测试的应用呢？接下来在设计用例的时候大家就会慢慢明白的。大部分人对上面场景中提到的功能，首先想到的是最基本的用例，如：

- 添加一个或多个联系人或号码进入黑名单，阻止通话。

- 从黑名单中删除一个或多个联系人或号码，阻止通话。

有点经验的测试工程师肯定会想到界面上的显示问题，那么就又出现了新的用例：

- 当黑名单未添加任何一个号码的时候，界面显示是否正常。
- 当黑名单添加了一个号码之后，界面显示是否正常。
- 当黑名单正好满一个屏幕的时候，界面显示是否正常。
- 当黑名单列表超过一个屏幕的时候，界面显示是否正常。

这里笔者也想多写点基本的用例，但是乍一看这个功能也的确只有这样一些基本的用例了。接下来我们就从生活场景中来获取一些新的用例设计思路吧。使用黑名单功能的原因基本有以下几种：

- 被某个陌生号码一直骚扰
- 有意识地不想和某号码联系
- 恶作剧？
- 女神号码？
- ……

也许还有许许多多的原因，这里就不细究了。接着我们继续分析一下，如果日常使用的话，会有哪些使用流程上的场景？或者会带来其他的测试用例设计切入点？试举一下：

- 直接添加一个号码到黑名单。
- 从联系人中选一个联系人添加到黑名单。
- 从通话记录中选一个号码添加到黑名单。

基本上用到的就是这 3 种。但是，单从添加号码这个行为上挖掘生活习惯还不够，还需要从另外几个不确定因素中，去深挖联系人和号码这两个因素，继而我们得到了：

- 联系人的存储方式（可能来自于 gmail 等帐号的同步、直接创建、从短信添加、从通话记录添加，甚至微信等应用的摇一摇添加等）。
- 号码的长度（固话、移动电话、短号码、小灵通等）。
- 号码的不同表现方式（比如 18621519900 和＋8618621519900 是同一个号码，包括漫游，甚至跨国号码等）。

最后还有一点也很重要，部分使用黑名单功能的用户是商务人士，这一点测试工程师是需要知道的。很多商务人士会使用双卡双待的智能机，这对于黑名单功能来讲也是一个至关重要的用例。

在测试的过程中，我们会遇见各式各样不熟悉的功能和需求。测试工程师也许不能像本例那样顺利地一步一步往下思考，但是测试活动本来就不是一个能够做到完美的行为，更多的是利用我们的态度、策略、知识、技能以求尽可能接近完美。通过在实际生活中去用这些应用，会让我们拓展思维，更贴近用户，更快地去接近这个完美。何乐而不为呢？

7.4　社交应用分层设计实践案例

前面 3 节并没有给出非常详细的测试用例设计方法，更多的是给大家提个醒，给个设计上的思路。在测试用例设计、探索性测试、随机测试等时候，无论我们觉得无从下手也好，找不到缺陷也好，一切原因归根结底都是来自于我们的无知。对于一切未知的东西，人天生会产生恐惧、迷茫，所以我们做测试的时候就必须去深入了解与被测对象有关的知识，这样才会看得更多、更深，思路也就会随之展开。



提示：从本节开始，笔者会举一些具体应用的测试用例设计和实现的方法。其中关于移动系统和开发语言本身的基础知识就不细说了，有需要的朋友可以自行谷歌。接下来我们看看下面这个“社交应用”的背景吧：

应用类型：社交应用

所在系统环境：Android，iOS，windowsphone 等

应用核心功能：发布自己喜欢的图片、评论、互粉等

应用所属结构：C/S

应用帐户：可使用自己的账号登录，也可使用新浪微博等第三方账号登录

现在假设项目开始了，我们作为测试人员开始接触需求，经过了解之后发现，这个应用与图片式的微博很相似。但是当我们开始设计用例的时候却发现，用例量远远超出我们的想象，这里列举一些功能核心的测试点：

- 不同账号的登录。
- 发布一张或者多张照片。
- 测试照片的不同来源。
- 测试照片或智能机的经纬度。
- 测试不同网络下的交互。
- 测试交友功能。
- 测试评论、喜欢、互粉等功能。
- 测试照片预览、放大、旋转等功能。
- 测试发布照片时候，@好友、增加描述、增加定位、增加滤镜等功能。
- 多智能机同时登录。
- ……

以上每一点都可以延伸出很多个测试用例。当我们辛辛苦苦正在编写测试用例的时候得知，原来给的测试时间只有不到 2 周，望着只有一两个人的测试队伍，已经欲哭无泪了。

此时，我们只有两个选择，其一告诉老板人手不够，时间不足，功能点太多做不完；其二是硬着头皮去做，发布之后出现各种缺陷。无论哪种选择都会降低老板对测试人员的好感。原因很简单，大家都希望事情能按时完成，克服困难并最终达到目的，谁也不愿意看到自己的项目一拖再拖。也许反馈的理由是客观存在的现实，但人手不够，时间不够也不是一时半会儿能够解决的问题。在移动互联网行业，无论公司大小，大部分项目的周期都差不多，开发与测试人员的比例都很悬殊（不得不说，这点我很想谴责很多人歪解了 Google 的 10：1 的意思，包括我老板）。其实随着项目业务的增加，单纯地增加人力肯定不是办法，所以我们需要有策略地去做测试。

很多朋友看到这里就想了，按照你这样的说法，人少、时间短，功能点又多又杂，还要按时完成，那么岂不是只剩下一个办法——自动化测试。说得没错。在移动互联网的项目中，测试必须要有自动化的部分。我这样说并非希望大家为了自动化而自动化，而是因为在项目中的确可以找到很多提升效率的方法，本例就是其中一个。此外还可以找到很多重复劳动的地方，比如界面的自动化，Android 渠道包的测试等。移动互联网行业人少、项目多的状况已经见怪不怪了，采用自动化是为了提升测试效率，帮助做回归测试，而不是很表面的认为是为了解放手工测试人员。不过很遗憾地告诉大家，在移动互联网中，无论自动化做成什么样子，手工测试的比例还是会很高。

要做到以上说的，测试人员深入了解产品是必不可少的。经过了解，我们发现这种社交应用大多是在客户端搭建一个框架，所有的图片、数据都是从服务器传递到客户端的。比如图片，都是经过服务器压缩过之后再传给客户端，客户端以 bitmap 的形式绘制出来，如下图所示。



其逻辑可以用下图来说明。



大约有 80%的数据是由服务器返回，这必然是一个非常重要的测试点。一般而言，我们会编写如下的测试点（由于是举例，简单写一下测试用例的步骤，请不要误解测试用例真正的编写方法）：

- step1：使用账号登录

  step2：发布一张图片

  step3：从我的界面或者好友的界面查看这张图片是否发布成功

- step1：使用账号登录

  step2：与某用户成为好友关系

对这些用例执行的时候有几种方法：

- 手动执行测试用例
- 界面自动化测试
- 分层测试

手动执行测试是现在大部分企业选择的方式，测试工程师苦不堪言，要在被压缩到极致的测试时间里执行完这些测试用例，基本上只有一种方法——加班。

界面自动化测试是现在大部分企业准备转型做尝试的方式，笔者个人不推荐在项目周期比较短的移动应用上做界面自动化测试。界面自动化测试更多地被应用在系统集成测试中。

本例中详细讲一下分层测试。既然我们已经分析判断出客户端上的数据大部分来自于服务器，与其将客户端交互逻辑、数据正确性、用户体验、客户端功能接口测试等全部集中在手工测试上，还不如将这些测试点都分开测试，一个点一个点地来验证，测试风险会降低不少。

假设与该应用对应的服务器有 3 个核心接口，分别是 Moment/pic_add、Moment/pic_update、Moment/get_feed，这里详细列了 Moment/pic_add 接口的参数表，如下所示：

```
Moment/pic_add
增加图片到一个组里并发布
URL
Http://monkey.test.test:12345
支持格式
JSON
HTTP 请求方式
POST
是否需要登录
是
请求参数
```

	必选	类型及范围	说明
Id	True	Int	为了区分每张图片的不同，每张照片都需要给一个 uuid
description	false	string	每张图片可以附带的描述
Lng,lat	false	Float	当前定位的经纬度

列出接口参数表之后，我们需要进行接口的实现。本例实现的接口测试方式并非仅为了测试单个接口，而是在测试单个接口的同时，紧紧地和应用业务做强绑定，从而将用户平时会使用到的功能逻辑一并进行自动化测试。不过这仅是针对与应用对应的服务器测试，与客户端并无联系。

我们可以在测试代码中创建第 1 个文件来实现所有的单独的接口，比如上面提到的 Moment/pic_add 的实现代码如下：

```
def Picture_add(self,id,description,lng,lat):
       fields = {
           'id':str(id),
```

```
            'description':str(description),
            'lng':str(lng),
            'lat':str(lat),
        }
#这里的 urlopen 是自己封装的，并非 urllib2 自带的方法
return self.urlopen('http://monkey.test.test:12345/Moment/pic_add
',fields)
```

一般情况下我们都是优先实现 Post 和 Get 这两种请求方式。接口的返回结果使用 json 解析，即可得到需要的参数数据。用同样的方法实现其余所有的接口。至此实现的是第 1 层，该层和每个单个的接口做强绑定。

然后我们创建第 2 个文件来实现关联性较强接口的组合。我们假设完成一个功能需要调用 Moment/pic_add 和 Moment/pic_update 这两个接口，那就需如下代码来实现：

```
def Picture_upload(self,description,lng,lat):
        picture_uuid = uuid.uuid1()
        Group_uuid = uuid.uuid2()
        Picture_add(picture_uuid,description,lng,lat)
        Picture_update(Group_uuid)
```

这一步实现的是第 2 层，往往是将第 1 层单独的接口中，关联较强或用户经常用到的接口组合封装成单独的方法。该层与应用的功能做强绑定，一个功能可能是由一个或者多个接口组成的。Picture_upload()就与应用中照片上传这个功能所对应。

第 3 层完全与业务强绑定。一个业务可能是由一个或者多个功能组成的。最后，利用 Python 的单元测试框架即可实现和业务强绑定的自动化测试。还是用上传图片功能来讲，最终的用例实现如下：

```
class MonkeyTestCase(unittest.TestCase):
    def setUp(self):
        self.user_login = True
```

```
    def tearDown(self):
        self.user_login = None

    def testUploadPic(self):
        Picture_upload(description,lng,lat)
```

用例的组织可以通过读取配置文件获取，根据自己的喜好进行更好地管理。这样，功能、业务测试就能够和接口测试一一对应起来了。

上传带经纬度、有描述的图片代码如下：

```
Picture_upload('test','35','60')
```

上传不带任何信息的图片代码如下：

```
Picture_upload('','','')
```

类似前面上传图片功能的测试用例还有很多可以写，当第 3 层封装好之后，只要添加好注释就可以让整个团队中的成员都参与到测试用例的设计和补充中来了。对于所有看到第 3 层的人而言，他们已经不必知道功能是什么、业务是什么、应用使用流程是什么了。他们只需要从各种类型的参数上去设计即可。

我们来总结一下这 3 层的逻辑，如下所示：

第 1 层：单一接口实现 ------------------ 接口文档

第 2 层：多个接口的组合封装实现 ------------------ 应用功能列表

第 3 层：多个方法的组合封装实现 ------------------ 应用业务列表



提示： 在我们测试这类应用的时候，往往都会有一些难以重现或不稳定重现的缺陷。由于导致缺陷的原因众多，不仅会给测试人员描述缺陷造成困难，也会给开发人员造成困扰。很多测试人员会说自己公司的开发人员主动性不强，老是说问题不是自己的，皮球踢来踢去，最后结果就是缺陷还是得不到修复。我们换位思考一下，其实很多时候出现这类情况的原因，是测试人员对于缺陷的定位不准确造成的。在抱怨别人之前，我们需要先审视自己。当业务强绑定的接口测试覆盖度足够高的情况下，一旦客户端发现数据上的缺陷，就可以第一时间排除是服务器的问题，那对于客户端的开发人员来讲，也可以更准确地定位问题从而修复缺陷，大家都是为了将产品做得更好，何乐而不为呢？

当服务器接口测试有了保证之后，对接下来客户端的测试就可以再细分，之后使用各种方法进行测试保证，例如：

- 界面自动化：可使用类似 Instrumentation 工具，来检查各个界面控件的位置、使用 MonkeyRunner 对每个界面做图形对比测试等。
- 内存测试：使用类似 Android 的 MAT、systrace 或 iOS 的 Instruments 针对某些内存消耗特别大的功能做测试。
- 代码规范：使用 Lint 和 FindBugs 等工具测试。
- 压力测试：客户端的压力除了用 Monkey 测试之外，还可根据需求自己编写相关压力测试的代码。
- 客户端接口测试：类似于服务器的接口测试，只是使用的框架不同。
- 单元测试：Mock 各种场景对客户端的代码做测试。

以上所列的都是比较粗的方向，更多的需要根据产品的特性、用户群和新功能实现等再做进一步的细分以及测试用例的编写。

本例的分享到这里就差不多的结束了，相信大家看完之后至少对在移动应用上分层测试有一个大概的了解。移动互联网的应用虽不如传统软件业一些项目来得庞大复杂，但是作为测试人员，对产品从需求到用户的了解比任何人都要清晰，同时需要很强的分析能力。别人看应用是一个应用，而我们测试人员看到的应该是应用的实现、结构、用户习惯等。这样才能帮助我们更好地测试一款应用。

7.5 联系人搜索案例测试设计实践

第 2 个案例就开门见山地来讲吧，先描述一下应用的背景。

应用类型：智能拨号软件

所在系统环境：Android，iOS，windowsphone 等

应用核心功能：联系人搜索，Voip，智能 IP 拨号

应用所属结构：纯本地应用

应用帐户：无账号

当下有很多的智能联系人搜索应用，如腾讯、360、触宝等。联系人应用是一个移动系统固有的应用，为什么用户还会有这么大的需求量呢？原因只有一个，智能机上的默认联系人应用太不智能。当你要查询一个想找的人、想在出差、出国等各种漫游的情况下得到优惠提示、想有很好的黑名单功能以免于骚扰，那么默认的联系人拨号应用几乎不可能实现。

每个智能联系人搜索应用的特色各有不同，但最常用的核心功能都一样——搜索。如何让用户在使用各种方式进行搜索的时候，都能尽快地查到自己想要找的人，就成了一个难题。现在假设测试人员要针对某款应用的搜索联系人功能做测试工作，首先我们得到的搜索策略如下表所示：

搜索匹配	输　入	匹配结果
全匹配	monkeychenye	monkey chen ye
	chenyemonkey	monkey chen ye
	yemonkeychen	monkey chen ye
正序首字母匹配	mcy	monkey chen ye
正序部分匹配	monchy	monkey chen ye
倒叙首字母匹配	ycm	monkey chen ye
倒叙部分匹配	ychmo	monkey chen ye
乱序首字母匹配	cym	monkey chen ye
	ymc	monkey chen ye
乱序部分匹配	chymo	monkey chen ye
	ymoch	monkey chen ye

了解完需求之后，我们还需要做一件事情——准备测试数据。这些搜索策略也许在某个联系人名字上测试通过了，但不能确保在其他各种形式的名字上都通过。所以，在真正测试之前先要设计测试数据。要做到这些，我们需要了解用户的需求，比如联系人姓名的显示形式：

- 全中文名称。比如“陈晔”、“移动测试会”。
- 全中文复姓名称。比如“欧阳振华”。
- 多音字姓氏名称。比如“单一平”。
- 全英文名称。比如“Peter Parker”。
- 中英文名称。比如“陈晔 Monkey”。

- 带有特殊字符的名称：比如“陈晔 ［移动测试］”。
- 全号码名称。比如“18621519900”

基本上也就是以上这些情况，再根据边界制做一定的设计，就会出现测试用例。可问题随之而来，一个应用的搜索功能就有那么多的测试用例，如果在项目时间和人力有保证的情况下，那么有多少用例都不怕，问题就在于现实中我们遇到的情况往往都是相反的。所以，很自然地就需要用自动化这个工具来帮助我们高效率地进行数据驱动的测试和回归测试。

本例中的自动化工具使用的是 Android 系统的原生测试框架 Instrumentation。如果要对搜索功能进行自动化测试，需要输入测试用例，通过使用应用的搜索方法之后，再对搜索列表做遍历查询，看看是否有期望的结果。所以需要先准备两个文本文档，一个存放测试用例（用户真实的搜索数据），另一个存放期望搜索出来的结果（用户想要的结果）。接下来看看如何使用 Instrumentation 框架吧。

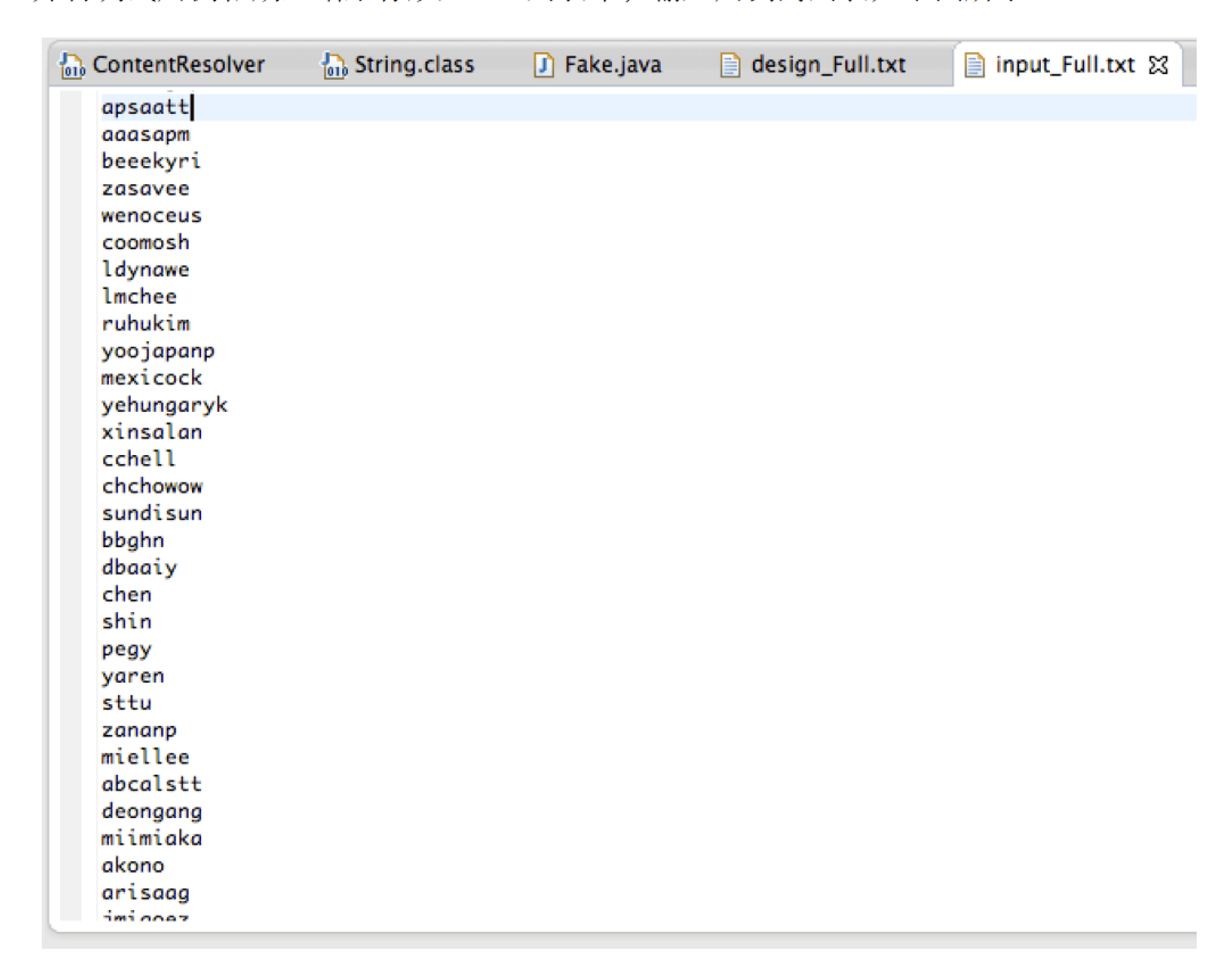
我们需要先新建一个 Android junit test 的项目工程，在 ContactTestCase 类中继承 ActivityInstrumentationTestCase2 类，定义相关测试的 Activity 名称等一些常规设置。核心代码如下：

```
//继承 Android 测试类
public class ContactTestCase extends
                    ActivityInstrumentationTestCase2<DialerActivity>

//新建实例，指定 Activity 名称和 Package 名称
          DialerActivity mActivity;//DialerActivity 是被测应用的核心类
          ContentResolver mCR;

          public ContactTestCase() {
                    super("com.contact.smartcontact", DialerActivity.class);
          }
```

接着我们需要将先前准备的测试用例数据和保存期望结果的两个文本文档放入测试工程的 assets 目录下，以方便在测试的过程中读取。在 setup()方法中将对象实例化并将测试用例和期望结果存放入 list 列表中，输入用例的列表如下图所示。



保存期望结果的文本文档内容如下图所示。

```
ContentResolver    String.class    Fake.java    design_Full.txt    input_Full.txt    »69
Ptolemaios celestial Nadleeh
Seravee seraphim Arios
Mothership CHERUDIM Being
Titans OCEANUS Apsaras
arios Seraphim Anance
being Nadleeh Kyrios
kyrios seravee Celestial
Qing Zhao Seravee
Wen KEVIN Oceanus
Cootek SONGSHAN Mothership
WENTING Luan Dyname
LEE Min China
KIM hyun RUSSIA
PARK yoo JAPAN
CHO KYU Mexico
KIM YE Hungary
Xie Vogins alan
Sabella Chen chen
Dan chow chow
Carter christina christina
Daniel SUN sun
Bin HUNQIN bcg
ayi Dong baba
chen Ayi dzx
kyu shin myung
pan peggy nana
yang SKY renee
alex SUTTON wu
ROBBIE Zhang pan
MITCHELL Lee peggy
ALEX SUTTON abc
```

核心代码如下：

```
//测试程序初始化
protected void setUp() throws Exception {
            super.setUp();
         mActivity = this.getActivity();
                 mCR = mActivity.getContentResolver();
                 try {
//获取在文本文档中的测试用例和期望结果
        InputStream Input_TestCase = this.getInstrumentation().getContext
().getAssets().open("Input_TestCase.txt");
         InputStream expectResult = this.getInstrumentation().getContext
```

```
().getAssets().open("exceptResult.txt");



BufferedReader reader_Test_Case = null;
BufferedReader reader_ExpectResult = null;

// String encoding="GB2312";
         reader_Test_Case = new BufferedReader(new InputStreamReader(Input
_TestCase, "GB2312"));
          reader_Expect_Result = new BufferedReader(new InputStreamReader
(expectResult, "GB2312"));

String text_input = null;
String text_expect = null;

for (int i = 0; i <= 30; i++) {
   if ((text_input = reader_Test_Case.readLine()) != null) {
         TestCaseList[i] = text_input;
     }
}
                  for (int i = 0; i <= 30; i++) {
                           if ((text_expect = reader_Expect_Result.readLine
()) != null) {
                           ExpectResultList[i] = text_expect;
}
                  }
                  } catch (FileNotFoundException e)
                  {
                           e.printStackTrace();
```

```
            }
            while (SmartDialerService.isDBIndexDone() != true) {
                Thread.sleep(2000);
            }}
```

以上代码的意思是，在每次执行全部用例前，先将测试用例和期望结果读取 30 个数据到特定的数组中，测试用例数组中的数据就是模拟用户搜索用户名的时候找到的数据，比如：

- wjl
- cy
- yidocsh

而在期望结果的数组中的数据则是用户输入搜索数据之后，想要得到的结果，对应上面的测试用例子，比如：

- 王佳梁
- 陈晔
- 移动测试

之后，我们需要做的就是将这些测试数据一个一个放入搜索方法里去，然后在搜索方法返回的结果列表中，去匹配期望结果数组中对应的数据是否存在。如果存在说明搜索方法功能正常，反之则说明搜索方法功能存在问题。我们来看一下这一步的核心代码：

```
        public int SearchRuslt(String N, String S) {
            String selection = DialerIndexProvider.NAME + "=" + "?" +
 " AND "+ DialerIndexProvider.NUMBER + "LIKE" + "%" + "?" + "%";
            String selectionArgs[] = new String[] {N};
Cursor c = mCR.query(DialerIndexProvider.CONTENT_URI, null, selection,sel
ectionArgs, null);
            if (c.getCount() > 0 && c.moveToFirst()) {
```

```
                    do {
                    String output = c.getString(NAME_INDEX);
                            if (output.equalsIgnoreCase(S)) {
                                    index = c.getPosition();
                                    break;
                            }
        } while (c.moveToNext());
            }
            return index;
    }
```

此方法中，参数 N 和 S 分别是传入的搜索数据和期望结果，如果能够遍历到相等的结果，那么返回该姓名在返回结果列表中的序号。现在我们已经能够对大量数据进行自动化测试了，但这仅能够判断应用的搜索功能在各种搜索条件下是否正常，而无法判断搜索的结果是否满足用户体验。也就是说，如果我们有 100 个用例，虽然全部通过了，但是每个结果返回值都在列表后面的 50 个当中。用户在智能手机上进行搜索之后，一页最多能显示 10 个结果，而他们往往懒到不愿意去翻页或者滚动，所以我们在测试的过程中，也需要非常关注搜索出来的结果排序是否靠前。

关于这个测试点，我当初在做项目的时候使用了比较偷懒的方法，具体看代码：

```
    public void testAbbreviations() {
            int total_count =0;
            int index = 0;
            for (int i = 0; i < TestCaseList.length(); i++) {
                    index = SearchRuslt (TestCaseList [i], ExceptRes
ultList [i]);
                    Log.e("+++", "index" + index);
                    assertTrue(String.format("i: %d input: %s outpu
t:%s", i,
```

```
                              TestCaseList [i], ExceptResultList [i]), index <
= 15);
          if index <= 5{
               total_count =total_count +1;
                   }else if index>5 &&index <=15{
               total_count = total_count +10;
          else{
               total_count = total_count +30;}
}
                   Result = total_count/TestCaseList.length()
          }
```

当搜索期望的结果在返回的列表中排在前 5 位、在 5 到 15 位之间或在 15 位以后，我们分别计数 1、10 和 30。最后用计数的总和去除以用例的数量，得出一个数字。如果每次测试执行完毕之后，在通过率一样的情况下，就能够从这个数字的增加或减少来判断该核心功能的体验好坏。

本例到这里也就结束了，我们来总结一下。每一个产品总有一些核心的功能，对这些核心功能做一些自动化或者压力、性能测试是大家都希望看到的。如果是一个系统平台，那么使用界面自动化做一些内置应用的启动操作等来保证回归测试肯定是必不可少。如果是一个应用的话，那么就可以像这个例子一样把一些核心功能抽出来做接口自动化，也能够起到同样的效果。

7.6 小 结

本章说了很多内容，虽然没有像传统软件测试用例设计方法，如“等价类”，“边界制”那么经典，但是我觉得在移动互联网行业做测试，更多的是要将这些传统的思想与自己的产品特性以及用户体验相结合，来思考如何设计。本章是我认为整本书最有价值

的一章，由于个人经验的关系，例子的数量和深入程度是有限的，但是包含的思路是无限的。大家可以沿着这些思路去总结适合自己的测试方法，从而更全面地去测试移动互联网的应用。

读书笔记

第 8 章　性能测试介绍和实践

其实最开始本章并不在编写计划中，但随着移动互联网应用的快速发展，各行各业的企业竟相展开了自己的移动项目，越来越多的测试工程师开始接触移动客户端的测试。让我奇怪的是，很多测试工程师对于所谓的“性能测试”这个概念的理解已经根深蒂固了，看到“性能测试”就马上联想到服务器测试。随即就出现很多问题：客户端有什么性能可以测试？客户端对应的服务器性能测试怎么做呢？等等。如果想要去进行移动互联网应用的性能测试的话，那就需要区分所谓的“性能测试”：一个是移动客户端应用的性能测试，另一个是服务器性能测试。从服务器角度来看，主要是并发量、吞吐量等性能，信息的来源可以是电脑、移动客户端、网页等，但无论是哪个，都不太影响服务器的性能测试。移动客户端应用的性能测试与服务器的性能测试不同，本章会举几个移动应用的性能测试点和工具予以说明。

8.1 Emmagee

现在针对各种各样的应用，市场上出现了很多对系统进行管理的应用，比如 360、手机管家等等。这类应用除了清理智能机系统垃圾之外，还会监控一些陌生来电或者电量流量等信息。这些应用普通用户平时使用自然没有问题，但测试工程师要测试这类信息的话，笔者不建议使用。虽然通过这类应用能够得到一些数据，但是它的逻辑其实就是一个黑盒——我们根本无法得知这类软件是如何获取信息的，这些未知来源的信息是无法作为测试数据的。



提示：笔者更建议测试工程师自己写一个工具来进行监控。Emmagee 是笔者曾经去网易参加沙龙时认识的一位好友孔庆云开源的一个 Android 性能测试工具。Emmagee 的 git 开源地址是：https://github.com/NetEase/Emmagee。这里简单介绍一下 Emmagee，有兴趣的朋友可以去看下具体的实现。

安装完毕之后，打开 Emmagee 会看到如下图（a）所示画面。

该界面会显示出目前系统中已经安装的应用。选择一个应用之后单击“开始测试”按钮，Emmagee 会自动启动被测应用，并在最上方显示流量、CPU、内存等数据，如下图（b）所示。

在该状态下可任意使用被测应用，Emmagee 会自动进行相关数据的记录，中止测试之后会生成相应的报告，并保存在 Android 目录中，如下图（c）所示。



（a）　　　　（b）　　　　（c）

提示：与其他性能测试工具相比，Emmagee 的使用方法简单易懂，能满足一定的需求。不过由于 Emmagee 诞生时间毕竟还短，自身还存在部分缺陷，所以需要大家一起来完善这个工具。更多相关的信息可以查看 Emmagee 作者的博客：http://kongqingyun123.blog.163.com/blog/static/63772835201302910540636/

8.2　Instrumentation

在第 6 章其实已经介绍过 Android 系统提供的原生测试框架 Instrumentation 了。该框架也可以用来做应用的性能测试。其实，使用 Android junit test 除了做单元测试之外，还可以进行功能测试、性能测试、压力测试等等。Instrumentation 毕竟是一个工具，怎么用还得视测试目的和测试切入点而定。Android 应用中使用 Instrumentation 展开性能测试的点很多，比如：

- 应用的多次启动时间

- 多个应用多次启动时间
- 应用内多个界面的切换时间
- 应用中 imageView 等控件对图像的绘制时间等
- ……

本节就 Android cts 中的 MultiAppStartupTest 测试类为例，来说明 Instrumentation 框架在性能测试上如何使用。

先创建测试类以及制定被测试应用的 Acitivity 和 Package 名称，核心代码如下：

```
public class MultiAppStartupTest extends InstrumentationTestCase {
private static final String PACKAGE_UNDER_TEST = "com.android.calculator2
";
    private static final String ACTIVITY_UNDER_TEST = "Calculator";
    private static final int ACTIVITY_STARTUP_WAIT_TIME = 1000;
```

启动一个应用需要将对应的 intent 作为参数传给 startActivity()，所以封装一个获取 intent 的方法也很重要，该例中的核心代码如下：

```
private Intent buildIntent(final String pkgName, String className,
boolean isMain) {
final String fullClassName = pkgName + "." + className;
Intent intent = new Intent();
intent.setClassName(pkgName, fullClassName);
intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
if (isMain) {
intent.setAction(Intent.ACTION_MAIN);
intent.addCategory(Intent.CATEGORY_LAUNCHER);
}
return intent;
```

```
}
```

用这个方法封装好之后，测试工作就简单了很多。在执行测试的时候先记录系统时间，然后再启动需要测试的应用，完全启动完毕之后再记录一个时间，将该时间与系统时间做一个减法，从而得出应用的启动时间。可能有朋友会问：在多少时间之内启动应用算是性能好呢？性能的好坏没有一个绝对的值，要根据不同的应用和测试功能点，来制定符合自己企业产品的标准。我们来看看最后计算启动时间的代码：

```
private long launchActivityUnderTest() {
long start = System.currentTimeMillis();
Intent i = buildIntent(PACKAGE_UNDER_TEST,
ACTIVITY_UNDER_TEST,
true);
Activity a = getInstrumentation().startActivitySync(i);
long end = System.currentTimeMillis();
long diff = end - start;
a.finish();
return diff;
}
```

运行 Android junit test 即可得到应用最后一次启动的时间。根据具体情况我们可以选择将结果打印在日志或保存在 SD 卡中等形式。

Instrumentation 在 Andorid 中的运用是非常广泛的，还能够 Mock 一些值和方法为单元测试提供方便。目前，大部分的界面自动化框架都是基于 Instrumentation 框架做二次开发封装而成的，说明它还是有很大的潜力可挖。

8.3 HPROF

HPROF 是一种后缀名为.hrpof 的文件，一个 heap dump 会保存为一个.hrpof 的二进制格式的文件。得到.hprof 文件的方法很多，在这里介绍两种。如果需要得到更精确的数据，可以在程序中加入 android.os.Debug.dumpHprofData() 方法，如下图所示。

--hprof	If set, this option will generate profiling reports immediately before and after the Monkey event sequence. This will generate large (~5Mb) files in data/misc, so use with care. See Traceview for more information on trace files.

第 1 种，相信很多朋友在使用 Monkey 工具时，仅仅就是设置一些单击参数和次数，其实 Monkey 工具只用一个参数就能够获取到 hprof 文件，这点很重要。

如同 SDK 文档中写的那样，我们可以在使用 Monkey 工具时增加--hprof 参数，随后就可以在/data/misc 目录中获取到 Monkey 测试开始前和结束时应用的两个.hprof 文件。

第 2 种，启动 DDMS 之后，选择需要测试的应用进程，单击 DDMS 左上角的按钮，即能够获取该应用在该时刻的.hprof 文件。

要分析 hprof 文件，可使用 Eclipse 的 MAT 插件进行解析（MAT 的插件可直接在 Eclipse 中的 Market 中找到）。我们对获取的 hprof 文件不能直接进行分析，必须从 Dalvik 格式转换成 J2SE HPROF 格式，否则会报错。转换的方法 Android SDK 中也已经提供了，如下图所示。

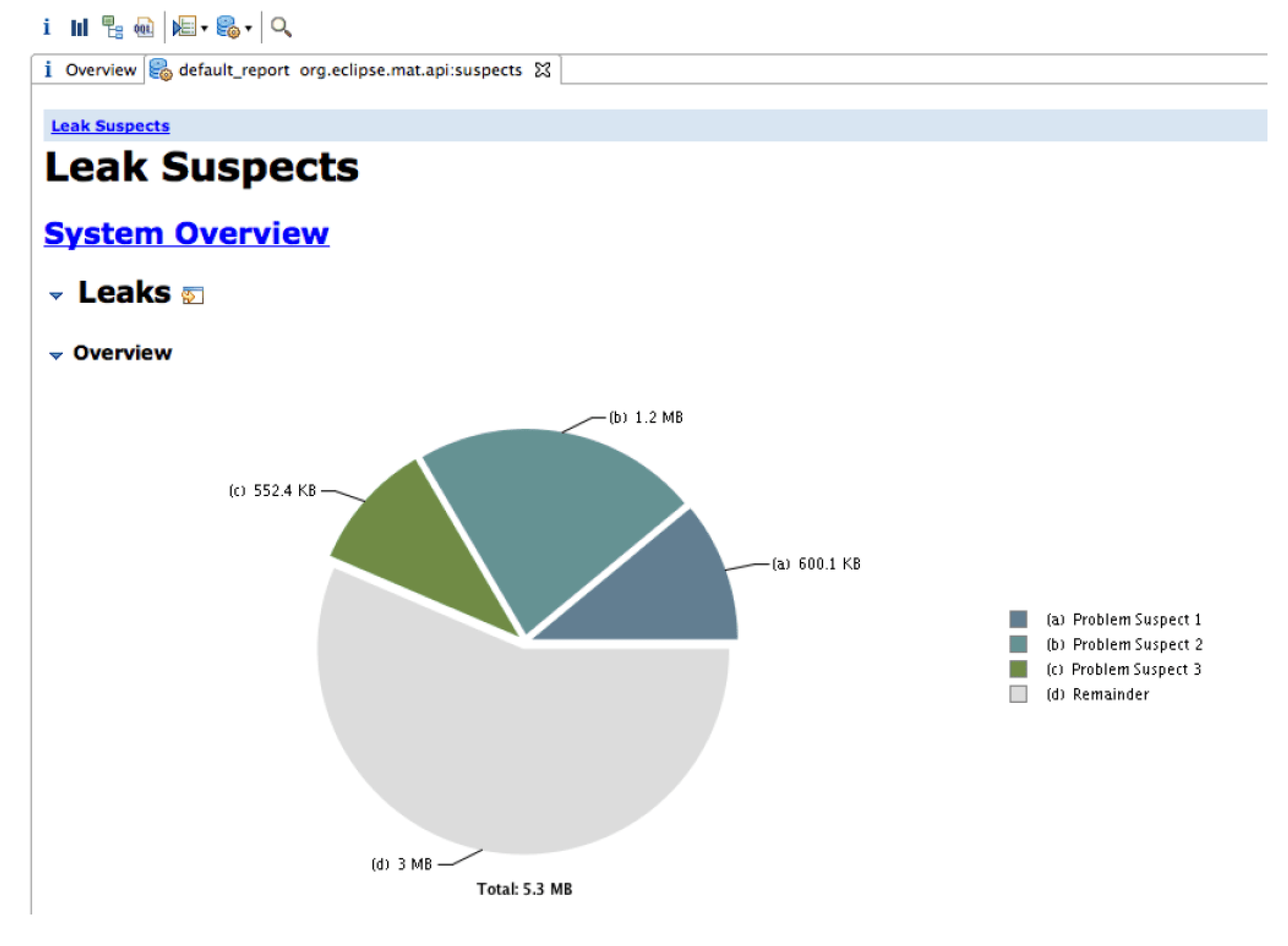
HPROF Converter

The `hprof-conv` tool converts the HPROF file that is generated by the Android SDK tools to a standard format so you can view the file in a profiling tool of your choice.

```
hprof-conv <infile> <outfile>
```

You can use "-" for `<infile>` or `<outfile>` to specify stdin or stdout.

可直接使用 hprof-conv <infile> <outfile>命令来进行文件转换。转换完成后即可使用 MAT 正常打开，会看到如下图所示的分析图。



具体问题可进入树状图进行分析。比如泄漏的问题一般情况下需要开发工程师一起分析，因为他们毕竟是最了解代码结构和方法的人。当然，这里并不是指测试工程师不

需要去了解源码，测试人员了解源码是为了了解原理，深入地了解功能，才能更好地设计测试用例。Android 中经常遇见有内存泄漏的点就是 bitmap，需要特别注意。



提示： 更多关于 MAT 的使用可关注官方的博客链接：http://memoryanalyzer.blogspot.com/ 。

Android 本身也提供了比较简单的命令来查看内存。

```
adb shell dumpsys meminfo <yourpakagename>
```

运行之后，可看到以下的日志：

```
applematoMacBook-Pro:tools apple$ adb shell dumpsys meminfo com.moregg.vida
Applications Memory Usage (kB):
Uptime: 193468205 Realtime: 193468205

** MEMINFO in pid 7585 [com.moregg.vida] **
                   Shared  Private     Heap     Heap     Heap
             Pss    Dirty    Dirty     Size    Alloc     Free
          ------   ------   ------   ------   ------   ------
  Native    3317      728     3260    12992     2865     1126
  Dalvik    6766     4828     6388     6388     6134      254
  Cursor       0        0        0
  Ashmem     942     1716       84
Other dev      4        0        0
 .so mmap    872     1760      228
.jar mmap      0        0        0
.apk mmap    297        0        0
.ttf mmap    448        0        0
.dex mmap   1876        0        4
Other mmap  1360       16       32
 Unknown     196       16      196
   TOTAL   16078     9064    10192    19380     8999     1380

 Objects
               Views:       52         ViewRootImpl:        1
         AppContexts:        3           Activities:        1
              Assets:        3        AssetManagers:        3
       Local Binders:       13        Proxy Binders:       20
    Death Recipients:        0
     OpenSSL Sockets:        0

 SQL
         MEMORY_USED:        0
  PAGECACHE_OVERFLOW:        0          MALLOC_SIZE:        0


 Asset Allocations
    zip:/data/app/com.moregg.vida-1.apk:/assets/Helvetica.ttf: 78K
```

其中的 HeapSize 是指总值，Heap Alloc 是实际的使用值，Heap Free 是剩余的值。其他参数可参考 Android SDK 中<Debug.MemoryInfo>的解释，如下图所示。

Fields		
public static final Creator<Debug.MemoryInfo>	CREATOR	
public int	dalvikPrivateDirty	The private dirty pages used by dalvik.
public int	dalvikPss	The proportional set size for dalvik.
public int	dalvikSharedDirty	The shared dirty pages used by dalvik.
public int	nativePrivateDirty	The private dirty pages used by the native heap.
public int	nativePss	The proportional set size for the native heap.
public int	nativeSharedDirty	The shared dirty pages used by the native heap.
public int	otherPrivateDirty	The private dirty pages used by everything else.
public int	otherPss	The proportional set size for everything else.
public int	otherSharedDirty	The shared dirty pages used by everything else.

8.4　Gfxinfo

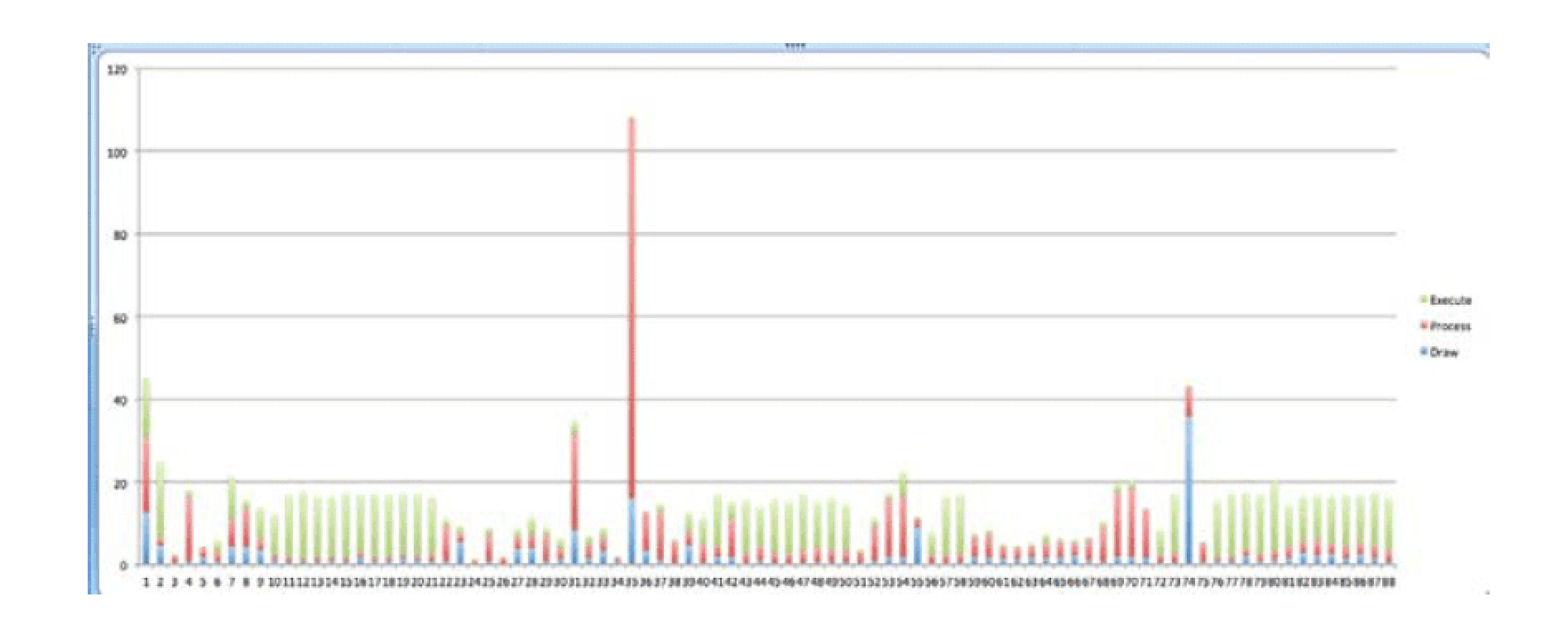
该工具是 Android4.1 新引入的，可以在开发者工具中看到新添加的一项“Profile GPU rendering”。如果在你的 Android4.2 设备上看不见开发者选项，可以在“关于手机”或者“关于桌面选择”的界面底部，单击“版本号”7 次。如下图所示。



勾选 Profile GPU rendering 选项之后，即可操作需要测试的界面。在终端运行下面这个命令：

```
adb shell dumpsys gfxinfo <your package name>
```

从产生的日志中可以看到有一段“Profile”的数据，是包含了 3 个数据的表。是我用 Excel 重新绘制了一下数据，使得查看效果更直观。



> 提示：关于图表的分析，这里摘录了“ImportNew 孙立”对相关文章的译文，翻译得非常好。
>
> 英文原文：Android Performance Case Study　编译：ImportNew - 孙立
>
> 译文地址：http://www.importnew.com/3784.html

以下是孙立的译文（我自己举例的图表中，数据的对应关系分别是：蓝色对应 Draw，红色对应 Process，绿色对应 Excute）。

每一列给出了每一帧花在渲染上的时间估计：

“Draw”是指 Java 层用在创建“display lists”（显示列表）上的时间。它表明运行例如 View.onDraw(Canvas)需要多少时间。

“Process”是指 Android 2D 渲染引擎用在执行“display lists”上的时间。你的 UI 层级（hierarchy）中的 View 数量越多，需要执行的绘画命令就越多。

“Execute”是指将一帧图像交给合成器（compositor）的时间。这部分占用的时间通常比较少

提醒：

要以 60fps 的帧率进行平滑地渲染，每一帧所占用的时间需要少于 16ms。

关于“Execute”：

如果 Execute 花费很多时间，这就意味着你跑在了系统绘图流水线的前面。Android 在运行状态时最多可以用 3 块缓存，如果此时你的应用还需要一块缓存，那应用就会被阻塞，直到 3 块中的一块缓存被释放。发生这种情况一般有两个原因：第 1 个原因是你的应用在 Dalvik(java 虚拟机)端画的太快，而在它的 Display list 在 GPU 端执行太慢。第 2 个原因是你的应用在前几帧的渲染上花费了太多的时间，一旦流水线满了，它就跟不上，直到动画完成。这些是我们想在下一个版本的 Android 改进的地方。

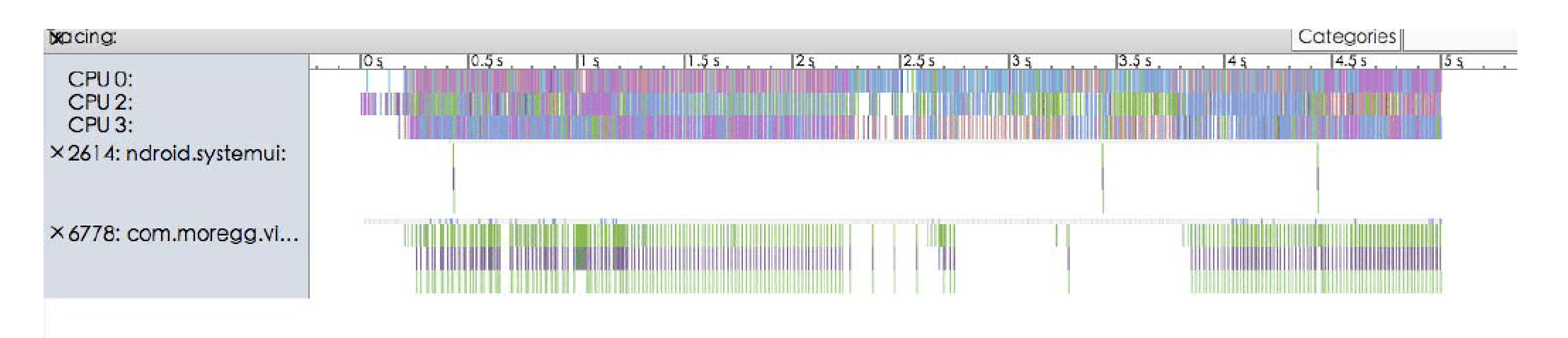
上图明显地证实了我的疑虑：该应用在大部分时间运行良好，但在某些时候会发生丢帧现象。”

8.5　Systrace

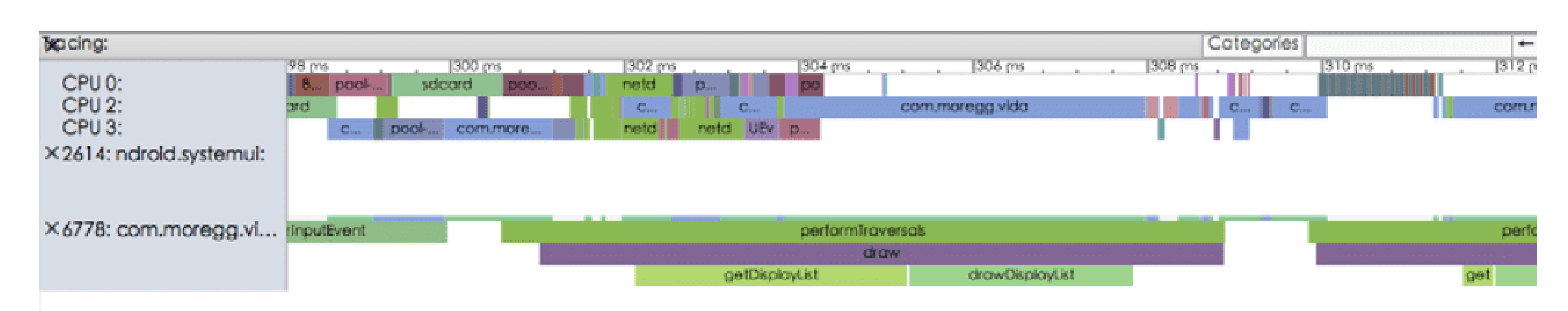
Systrace 同样只支持 Android4.1 及以上版本。它能够让我们在非常详细的时间轴上更清楚地看到应用和系统在性能上消耗的时间。我们可在 Android SDK 的 /tools/systrace 目录下找到 systrace.py 文件。

在执行脚本之前，我们需要在开发者工具下面选择需要记录 trace 的模块。

在终端输入 python systrace.py 即可运行程序，程序默认是抓取 5 秒内应用和系统的消耗，同时生成 trace.html 测试报告，打开报告可看到如下图所示的报告。



html 格式的报告可通过键盘上的“W”“A”“S”“D”4 个按键进行放大、缩小和左、右移动。放大之后，能够帮助我们更好地寻找性能瓶颈。





提示：关于该工具我自己也还没有很深入地去了解，所以没有写更多的实践。Systrace 支持更多的参数执行和功能，详情可在 http://developer.android.com/tools/debugging/systrace.html 查看。

8.6 TraceView

TraceView 是在 Android SDK 中自带的一个性能测试工具，可以在 tools 目录下找到。TraceView 能够很准确地查看到代码的更改给性能带来的变化。下面来看一个简单的例子，读者就能了解到 TraceView 的强大之处。

我们选择 Android 自带的 NotePad 应用工程来看，在 NoteEditor.class 中可以看到如下 onDraw()方法。

```
@Override
protected void onDraw(Canvas canvas) {
    int count = getLineCount();
    Rect r = mRect;
    Paint paint = mPaint;
    for (int i = 0; i < count; i++) {
        int baseline = getLineBounds(i, r);
        canvas.drawLine(r.left, baseline + 1, r.right, baseline + 1,
                paint);
    }
    super.onDraw(canvas);
}
```

我们在 NoteEditor.class 的 onCreate()方法中增加 Debug.startMethodTracing("NoteEditor");

同样的在 onDestroy()方法中增加 Debug.stopMethodTracing();

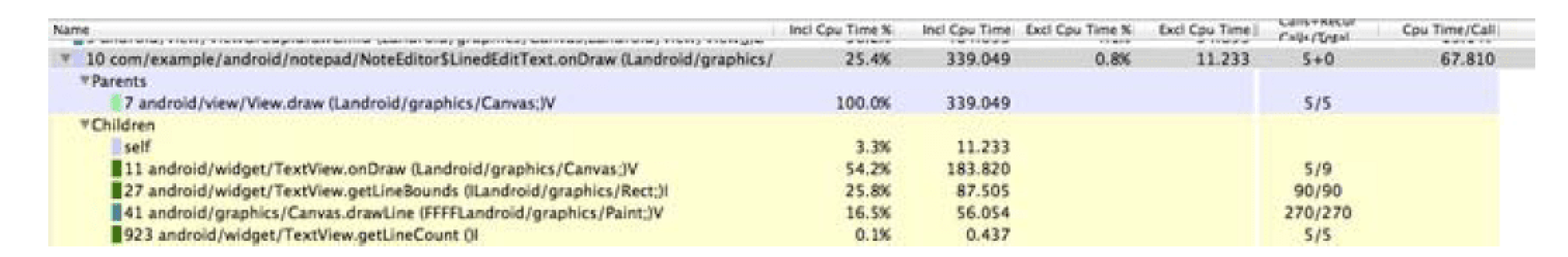
我们运行并使用一些简单的功能之后，可以在/sdcard/下找到 NoteEditor.trace 文件，然后在命令框中输入：

```
Traceview NoteEditor.trace
```

在 traceview 打开文件之后，我们可以看到如下显示。

Name	Incl Cpu Time %	Incl Cpu Time	Excl Cpu Time %	Excl Cpu Time	Calls+Recur Calls/Total
▼10 com/example/android/notepad/NoteEditor$LinedEditText.onDraw (Landroid/graphics/Canvas;)V	31.3%	1873.123	0.3%	18.169	88+0
▼Parents					
7 android/view/View.draw (Landroid/graphics/Canvas;)V	100.0%	1873.123			88/88
▼Children					
self	1.0%	18.169			
11 android/widget/TextView.onDraw (Landroid/graphics/Canvas;)V	84.3%	1579.305			88/92
50 android/widget/TextView.getLineBounds (ILandroid/graphics/Rect;)I	10.9%	204.177			205/205
124 android/graphics/Canvas.drawLine (FFFFLandroid/graphics/Paint;)V	3.3%	61.637			205/205
431 android/widget/TextView.getLineCount ()I	0.5%	9.835			88/88

我们能够看到 onDraw()及子方法调用的顺序和消耗，CPU 总消耗是 31.3%。继续我们的实验吧。在 onDraw()方法中尝试添加多个 canvas.drawLine()方法，然后可以看到如下的变化。



Name	Incl Cpu Time %	Incl Cpu Time	Excl Cpu Time %	Excl Cpu Time	Calls+Recur Calls/Total	Cpu Time/Call
10 com/example/android/notepad/NoteEditor$LinedEditText.onDraw (Landroid/graphics/	25.4%	339.049	0.8%	11.233	5+0	67.810
Parents						
7 android/view/View.draw (Landroid/graphics/Canvas;)V	100.0%	339.049			5/5	
Children						
self	3.3%	11.233				
11 android/widget/TextView.onDraw (Landroid/graphics/Canvas;)V	54.2%	183.820			5/9	
27 android/widget/TextView.getLineBounds (ILandroid/graphics/Rect;)I	25.8%	87.505			90/90	
41 android/graphics/Canvas.drawLine (FFFFLandroid/graphics/Paint;)V	16.5%	56.054			270/270	
923 android/widget/TextView.getLineCount ()I	0.1%	0.437			5/5	

明显可以看出 drawLine()方法占用率从 3.3%增长到了 16.5%。

接着我们还原修改，尝试给记事本增加点背景颜色，所以我们在 onDraw()方法中增加 canvas.drawColor(Color.BLUE);

然后可以看到如下变化。

Name	Incl Cpu Time %	Incl Cpu Time	Excl Cpu Time %	Excl Cpu Time	Calls+Recur Calls/Total
10 com/example/android/notepad/NoteEditor$LinedEditText.onDraw (Landroid/graphics/Canvas;)V	42.4%	3270.929	1.5%	114.846	38+0
Parents					
5 android/view/View.draw (Landroid/graphics/Canvas;)V	100.0%	3270.929			38/38
Children					
self	3.5%	114.846			
11 android/widget/TextView.onDraw (Landroid/graphics/Canvas;)V	46.4%	1517.457			38/42
27 android/widget/TextView.getLineBounds (ILandroid/graphics/Rect;)I	23.2%	758.809			736/736
26 android/graphics/Canvas.drawColor (I)V	22.6%	740.202			2944/3022
86 android/graphics/Canvas.drawLine (FFFFLandroid/graphics/Paint;)V	4.1%	135.154			736/736
620 android/widget/TextView.getLineCount ()I	0.1%	4.461			38/38

从报告中可以看出整个 onDraw()方法的 CPU 占用率急剧升高。

从 demo 中的 traceview 可以看到比较细颗粒度的修改，报告中相关名称的解释如下：

Exclusive：同级函数本身运行的时间。

Inclusive：除统计函数本身运行的时间外，再加上调用子函数所运行的时间。

Name：列出所有的调用项，前面的数字是编号，展开可以看到有些有 Parent 和 Children 子项，就是指被调用和调用。

Incl: Inclusive 时间占总时间的百分比。

Excl：执行占总时间的百分比。

Calls+Recur Calls/Total：调用和重复调用的次数。

Time/Call：总的时间(ms)。

如果在运行 trace 命令的时候碰见如下信息：

```
Failed to read the trace filejava.io.IOException: Key section does not hav
e an *end marker
```

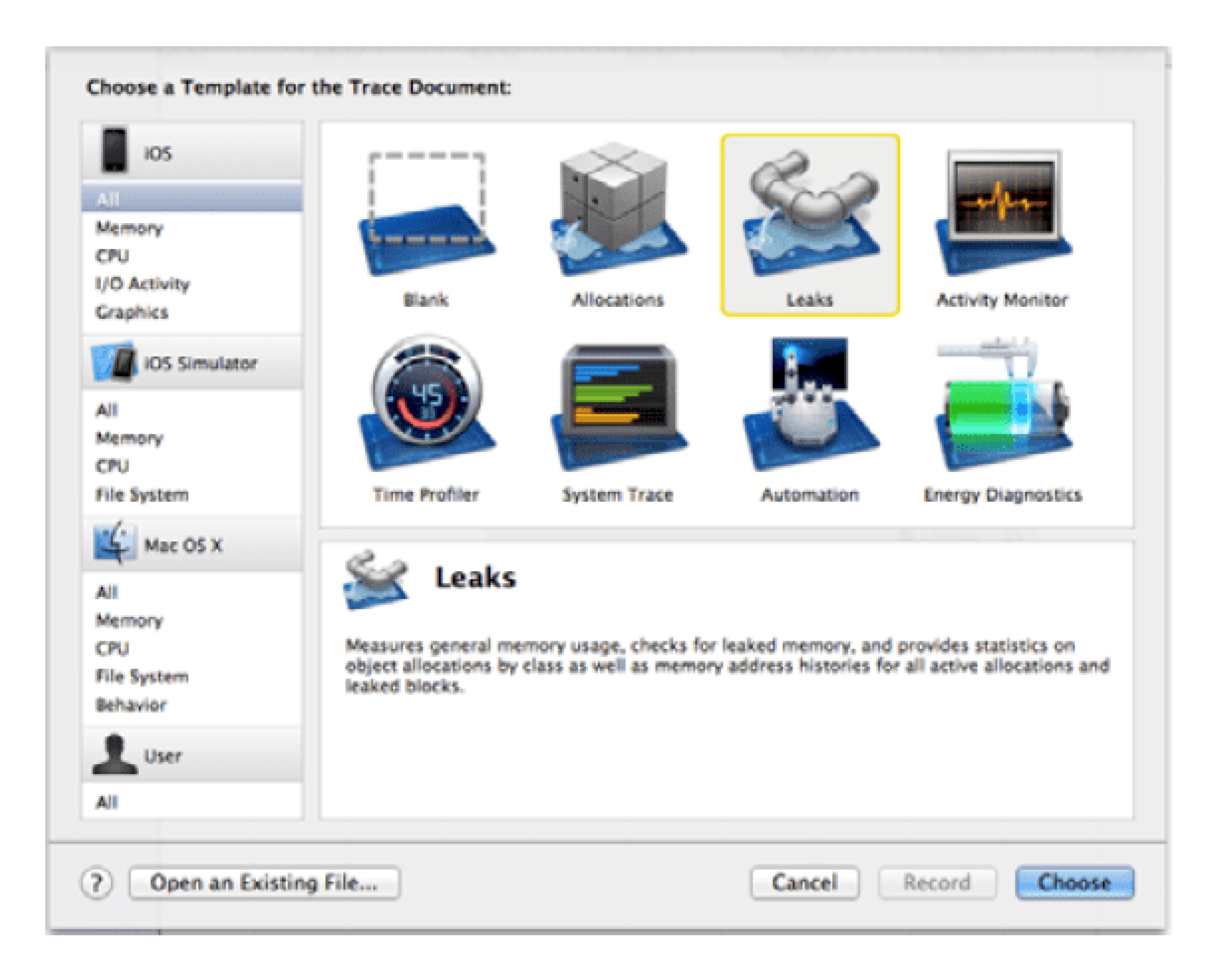
```
    at com.android.traceview.DmTraceReader.parseKeys(DmTraceReader.java:4
20)
    at com.android.traceview.DmTraceReader.generateTrees(DmTraceReader.ja
va:91)
    at com.android.traceview.DmTraceReader.<init>(DmTraceReader.java:87)
    at com.android.traceview.MainWindow.main(MainWindow.java:286)
```

原因是 trace 文件长度是 0 字节，需确认运行了添加有 Debug.stopMethodTracing(); 的方法。

8.7　Instruments——Leaks

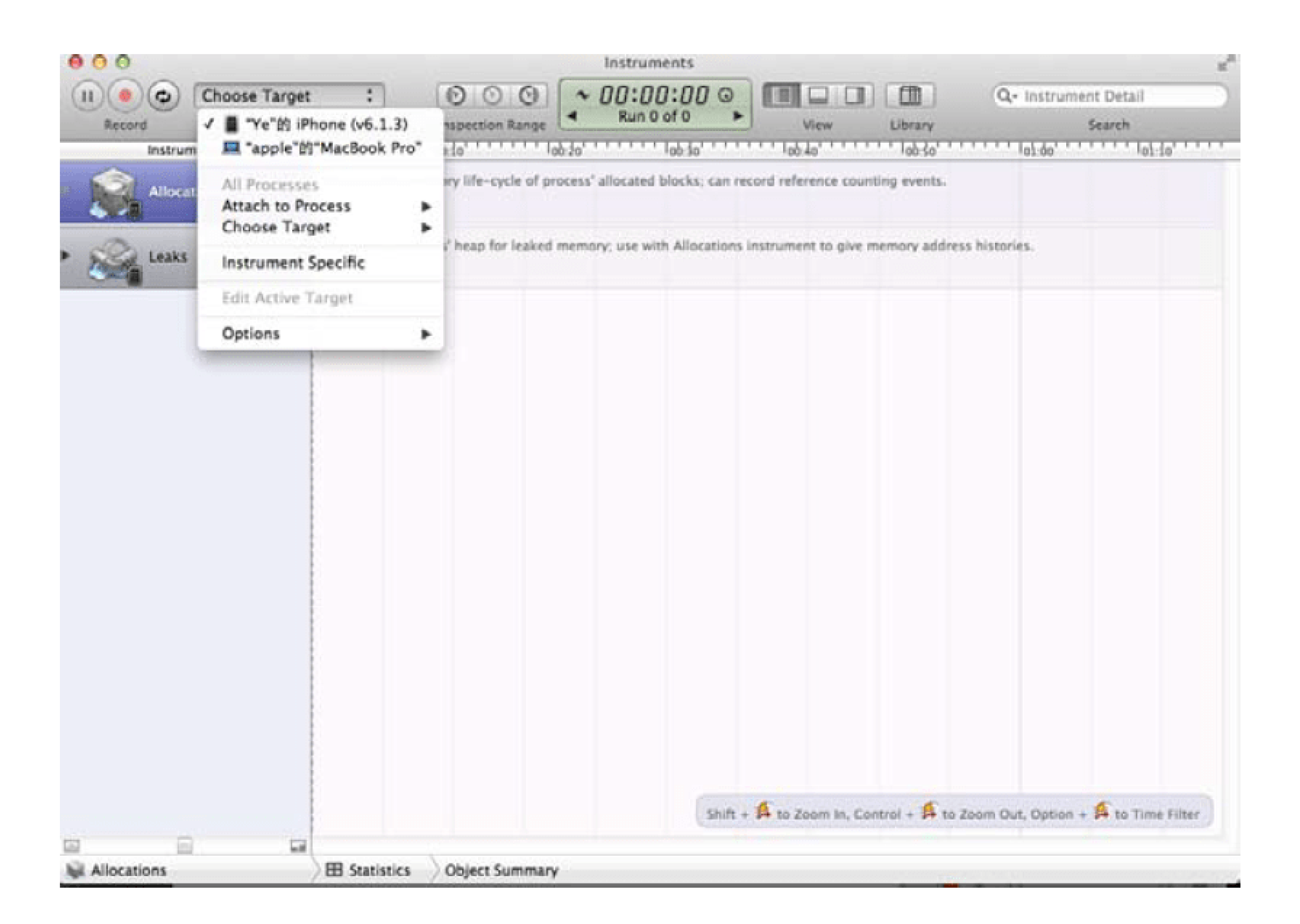
在第 6 章中介绍过使用 xcode 的 Instruments 做 iOS 智能机的界面自动化测试。Instruments 的功能很强大，本节再来介绍其另外一个常用功能——Leaks。

（1）打开 Instruments 之后可以看下如下界面。选择 Leaks。

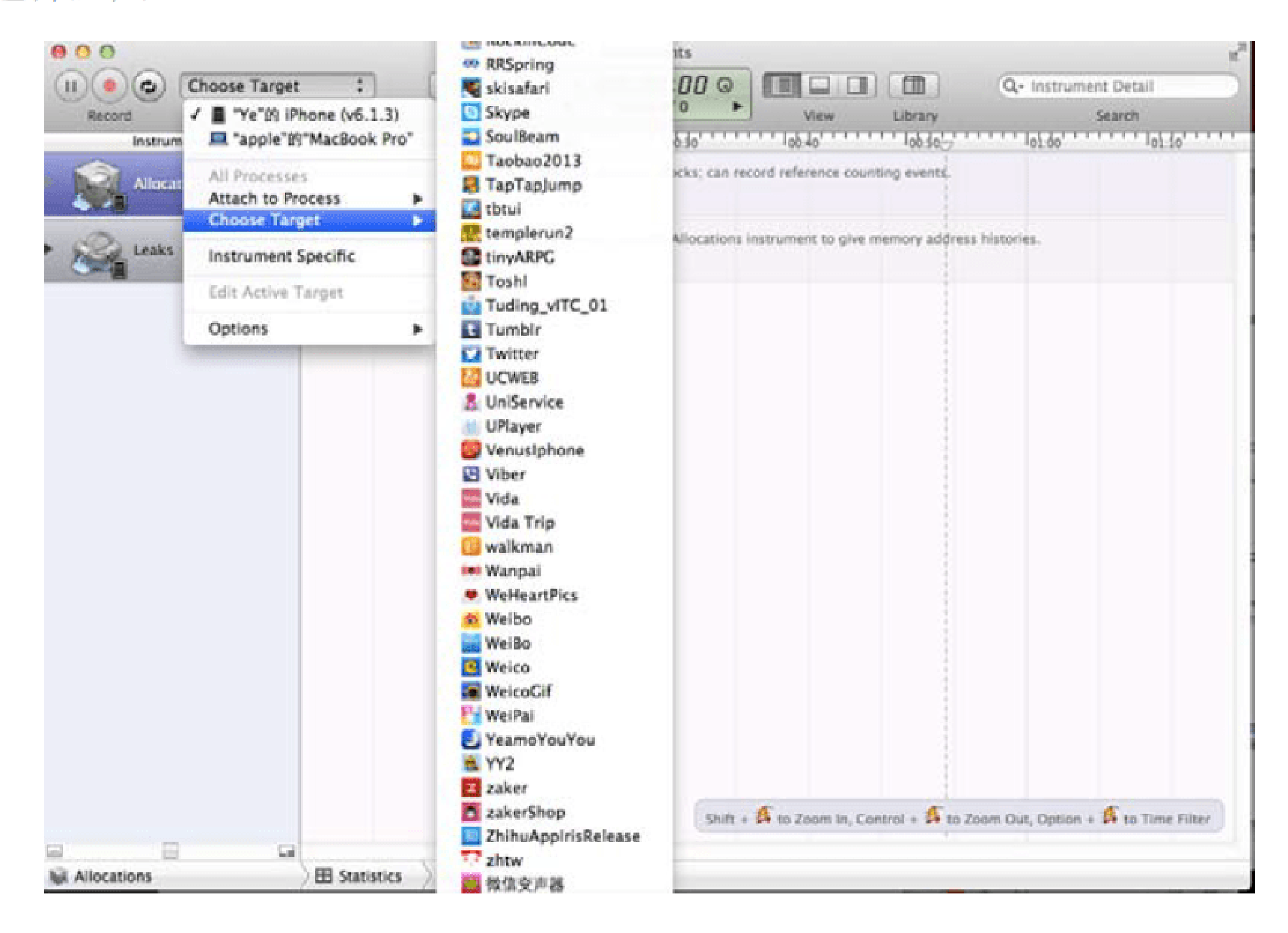


（2）随后选择需要执行分析的智能手机设备以及被测试的应用，本例选择的是 iPhone4 以及被测应用 Vida。

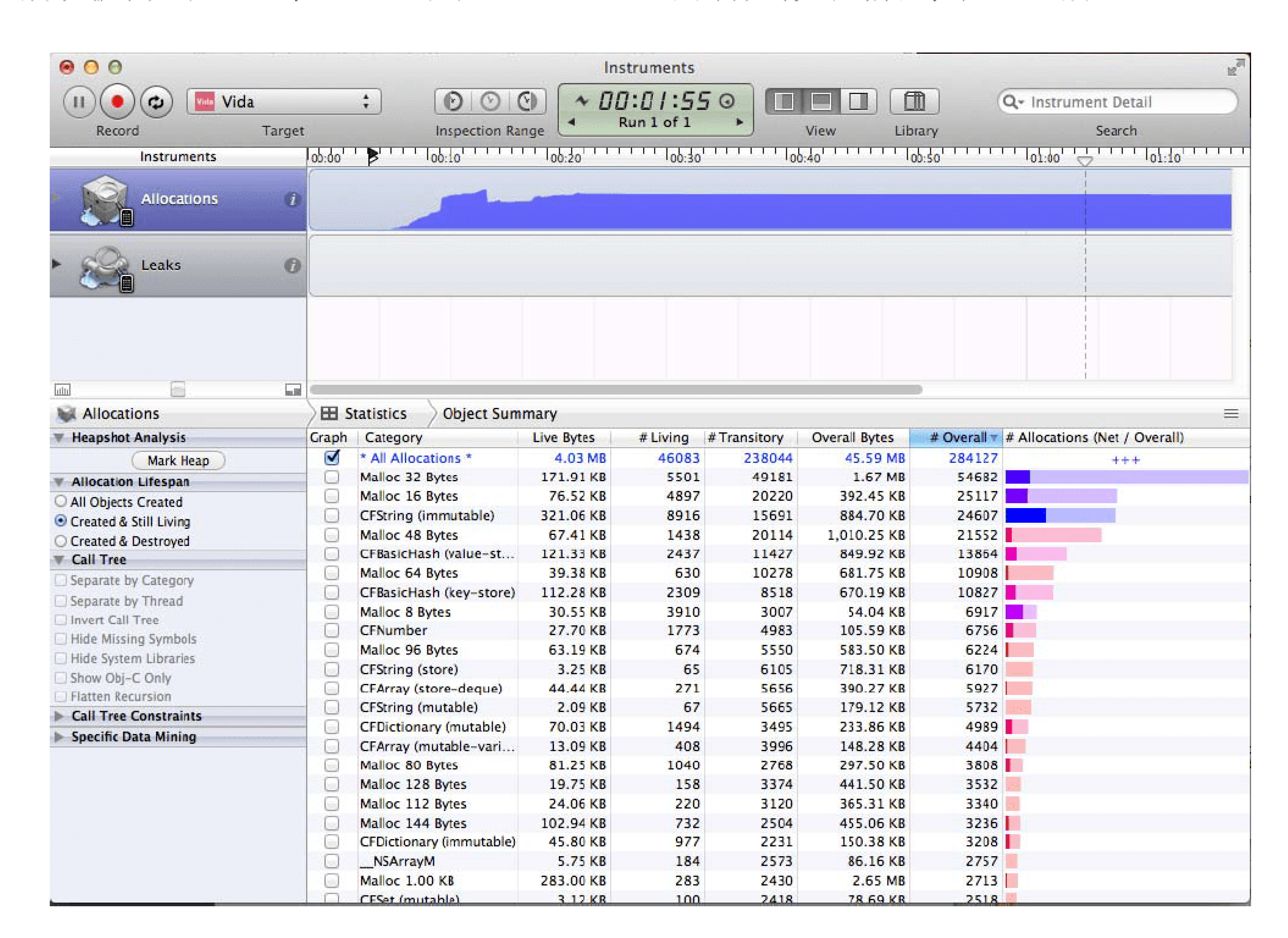
先选择了 Ye 的 iPhone，如下图所示。



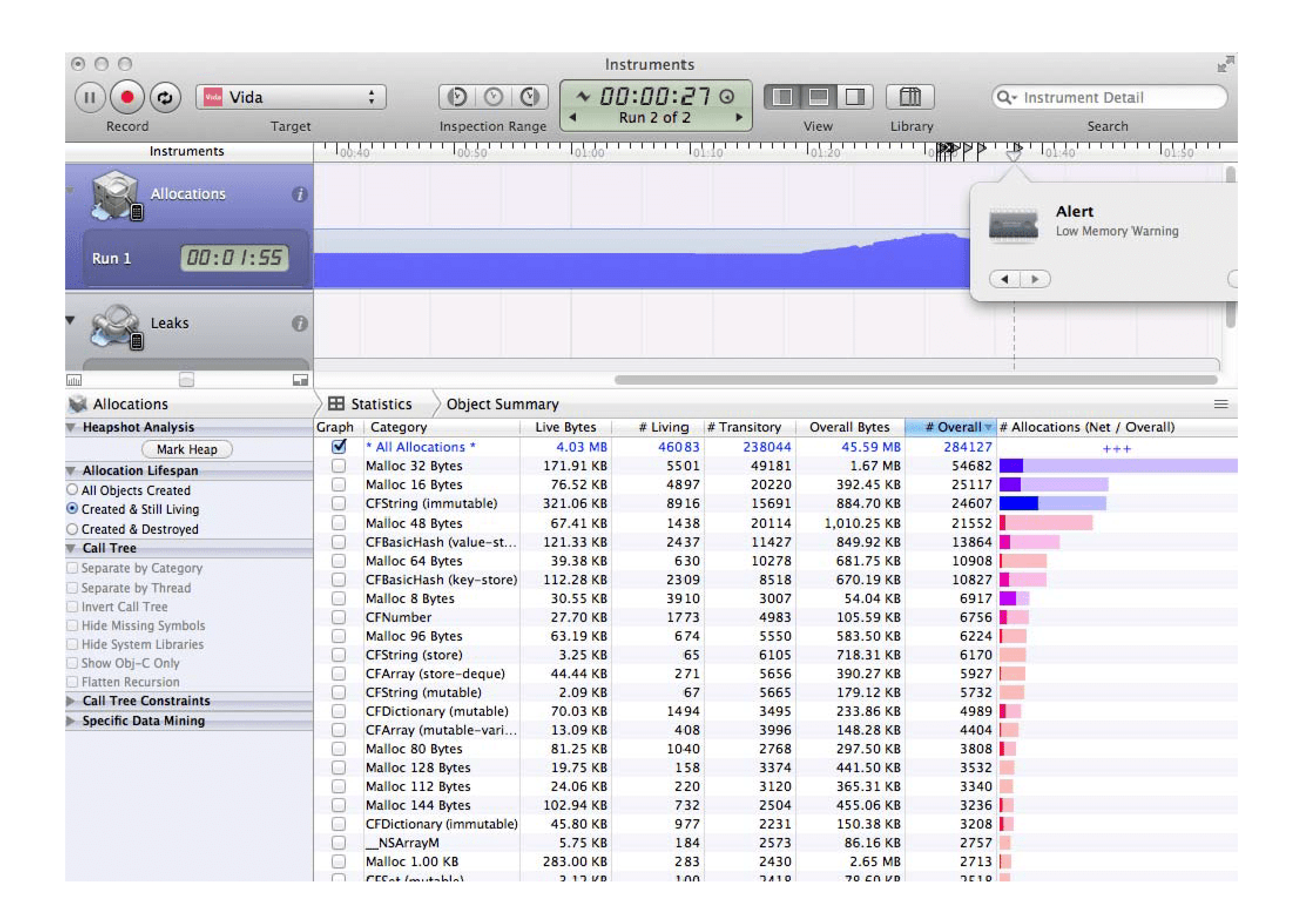
再选择应用 Vida。



（3）全部设置完毕之后，可单击左上角的 Record 开始测试，所选择的设备会自动启动被测应用进程，可一边使用一边查看整个内存的变化情况，如下图所示。



从上图显示的内存变化可以看出，虽然应用时占用了不少内存，但是一段时间之后，其占用量就形成了相对稳定的走势。此时，如果进行若干操作，比如打开应用中自带的相机则会看到明显的内存增长以及 Instruments 给出的内存不足的警告，如下图所示。



通过 Leaks 工具能够找出促使内存持续高涨的蛛丝马迹甚至直接原因，如果在调试的过程中应用直接崩溃了，对测试工程师而言自然是好事，因为从 xcode 中的崩溃日志就能方便快速地寻找到导致崩溃的源头。但是一般没有那么顺利，所以就需要借助像 Leaks 这样的辅助监控工具来一步一步接近源头。

8.8　Android 多分辨率自动化实践

Android 系统多分辨率的自动化测试其实是我很早就想做的，但由于各种原因拖到很晚才去实践。Android 的测试在真机上做的结果最为真实，但在测试时往往面临如下问题：

- 公司机器不够；
- 真机没有 root 权限；
- 大规模真机测试容易出现 adb 服务断开或者各种不可预计的问题。

所以我考虑的结果是，真机的测试肯定是要做的，但是在模拟器上实施自动化测试会简单方便很多，虽然结果的真实性有所欠缺，但总比没做好，路总是要一步一步去走的。

运行在模拟器上的测试方案要用到 Android Emulator、Python、Shell、MonkeyRunner、Instrumentation。Python 负责总体集成：Shell 脚本控制启动和关闭不同分辨率的 Emulator；MonkeyRunner 负责模拟非应用的操作以及协助截图；Instrumentation 负责应用内的操作；最后由 Python 进行图片对比测试。

（1）首先需要创建好各种分辨率的模拟器。这里我们先创建两个，分别叫做 Test1 和 Test2。接着我们来看一下启动模拟器的 Shell 脚本：

```
#!/bin/bash
# 指定 port，可以不干扰其他正在运行的模拟器。根据 sdk 文档，port 可以在 5554 至 5584 之间。
portNumber=5554
emulatorPID=`ps -ef | grep "emulator" | grep "port $portNumber" | awk '{print $2;}'`
if [ -n "$emulatorPID" ]; then
        echo "close emulator window"
        kill -9 $emulatorPID
fi

# 创建模拟器，force 选项能覆盖已存在的模拟器，要注意的是，sdcard 上的内容也会抹掉。$1 是可以在启动 shell 的时候从外部传递的参数变量
echo $1
```

```
# 启动模拟器，port 选项指定 tcp 端口。&是另起进程执行
echo "start avd"
emulator -avd $1 -port $portNumber&

echo "check emulator ready..."
emulatorSerialNumber="emulator-$portNumber"
adb kill-server
ready=`adb devices | grep "$emulatorSerialNumber"`
sleep 5
if [ -z "$ready" ]; then
        echo "emulator can NOT launch."
        exit 3
else
        ready=`adb devices | grep "$emulatorSerialNumber.*offline"`
        if [ -n "$ready" ]; then
                echo "wait-for-device"
                adb kill-server
                sleep 5
        fi
fi
```

（2）主 Python 中添加 StartDevice 方法：

```
def startDevice(emulatorName):
        os.system("./startEmulator.sh %s"%emulatorName)
```

我在启动模拟器之后碰见一个问题——模拟器初始是锁屏的，试用 MonkeyRunner 进行解锁，代码如下：

```
'''
```

```
from com.android.monkeyrunner import MonkeyRunner,MonkeyDevice
from com.android.monkeyrunner.easy import EasyMonkeyDevice
from com.android.monkeyrunner.easy import By
import sys
#from com.android.hierarchyviewerlib.device import ViewNode

Coordinate_list = []

"连接设备"
device = MonkeyRunner.waitForConnection()
easy_device = EasyMonkeyDevice(device)

#以 320*480 为标准算出坐标的比例从而扩展到各个分辨率中。
MonkeyRunner.sleep(5)
x1 = 50/float(320)
x2 = 250/float(320)
y1 = 350/float(480)
#让 Python 启动的时候从外部获取参数变量
for arg in sys.argv:
        Coordinate_list.append(arg)

print int(float(Coordinate_list[1])*x1)
print Coordinate_list[2]



device.drag((int(float(Coordinate_list[1])*x1),int(float(Coordinate_list
[2])*y1)),(int(float(Coordinate_list[1])*x2),int(float(Coordinate_list
[2])*y1)),1.0,10)
```

（3）解锁之后我们需要重置应用，Python 调用 os.system 进行卸载安装：

```
os.system("adb uninstall com.xxx.xxx")
os.system("adb install xxx.apk")
```

（4）如应用本身需要操作（比如需要登录，或者别的一些不跨 Activity 的操作），我这里调用 Instrumentation 启动 Android Junit Test，代码如下：

```
def AppLogin():
     os.system("adb shell am instrument -w -e class com.xxx.test.testClass
 com.xxx.test/android.test.InstrumentationTestRunner")
```

（5）然后是启动应用的各个界面对应的 Activity，MonkeyRunner 代码如下：

```
# -*- coding: utf-8 -*-

import os
from com.android.monkeyrunner import MonkeyRunner,MonkeyDevice
from com.android.monkeyrunner.easy import EasyMonkeyDevice
from com.android.monkeyrunner.easy import By
import sys

Coordinate_list = []

device = MonkeyRunner.waitForConnection()
easy_device = EasyMonkeyDevice(device)
#调用 python 的时候从外部传递参数变量
for arg in sys.argv:
         Coordinate_list.append(arg)

def screenshot(Activity):
```

```
        package = 'com.xxx.xxx
        activity = 'com.xxx.%s'%Activity
        print activity
        runComponent = package + '/' + activity
        device.startActivity(component=runComponent)
        MonkeyRunner.sleep(8)
        result = device.takeSnapshot()
        result.writeToFile('/Users/monkey/Pictures/screenshot/%s_%d*%d.p
ng'%(Activity,int(Coordinate_list[1]),int(Coordinate_list[2])),'png')


screenshot('Activity1Name')
screenshot('Activity2Name’)
```

（6）最后通过 Python 的 PIL 模块对比图片产生的结果，有的朋友会问：onkeyRunner 本身就带图片对比的方法，为什么不用？onkeyRunner 自带的方法其参数只能是 0-1 的之间的数字，由于 Android 屏幕截图中包含了一些不可控信息（比如电量、时间等信息），所以参数设置为 1 肯定是 false，而设置为 0.9，就算结果是 true 也很难得出一个具体的测试标准。所以我换用 PIL 进行对比。代码如下：

```
#sudo pip install PIL
def pil_image_similarity(filepath1, filepath2):
    from PIL import Image
    import math
    import operator
    image1 = Image.open(filepath1)
    image2 = Image.open(filepath2)
    h1 = image1.histogram()
    h2 = image2.histogram()
```

```
    rms = math.sqrt(reduce(operator.add,  list(map(lambda a,b: (a-b)**2, h1, h2)))/len(h1) )
    return rms
```

使用该方法，如果两张图的相似度是 100%，其结果是 0，数字越大说明相差得越大。

最后在总体集成的 Python 文件中将这些方法串起来，调用时传递的参数就是模拟器的名字以及分辨率数值就可以了。就如我一开始说的，也许模拟器的结果可信度不高，但至少可以在各个分辨率下查看应用在每个界面下的显示是否正常。如果不放心第一遍 MonkeyRunner 截下来的图。可以人工再检查一遍，作为以后测试的期望值。

8.9 小　结

本章对 Android 和 iOS 应用的性能测试做了介绍，并给出了实践。

在我参加各种沙龙的过程中，经常遇到一些朋友问：移动互联网应用去做性能测试真的那么重要吗？或者真的有意义吗？

我回答如下：

“移动互联网测试当前的现状是：测试人员和测试时间根本不够用，在这样的环境下，我们必须用有限的时间去尽可能地营造各种用户在将来几年内可能发生的使用场景。而应用的性能正是一个在短期不会爆发的问题，但对于测试人员而言，我们必须去保证应用在用户手中是长期稳定的，同时不会对用户的流量、手机性能等造成很大影响。”

读书笔记

附录 A　测试人员的自我修养（吐槽篇）

其实，长久以来有一句话一直积压在我心中，“软件测试工程师”真的是一个很神奇的岗位。该岗位的人在不同公司、团队、领导下所看到的、感受到的和学习到的可能是完全不同的东西。软件测试在国内发展时间虽然没有开发那么长，但是毕竟也有所积累。软件测试活动本身并不是昙花一现的工作，更注重的是长期积累总结、持续改进。在这里我并不是只关注软件测试工程师这样一个岗位的人，而是所有在做软件测试工作的从业人员，我们的学习需要有好的方法支持，需要长时间地积累。本章我总结了自己的一些感悟，希望大家从中能够提炼出自己的一套学习方法。

A.1 学会提出和解决问题

提问和回答可以说是人们经常做的两件事情。别的行业暂且不说，就测试行业来说，一个不会提出问题的人肯定不是一个好的测试人员。提问基本上分成以下几类：

- 不经过思考的提问。这类问题基本上都是××软件怎么安装、为什么这个日志看不懂等。
- 经过短暂思考的提问。经过短时间的思考就提问，自己不会主动寻求解决方案。
- 对于某问题思考之后。为寻求别人的建议或看法的提问。
- 自己经过思考。努力寻求解决方案无果，从而寻求别人的帮助。

不管是以上哪一类，我们都需要学会提问的方式。就拿 Android 自带的 Monkey 工具而言，以上几类提问方式被映射到实际的人身上会是以下这样：

- 有人知道 Monkey 怎么用吗？
- 有人知道 Monkey 怎么启动吗？
- 我使用 Monkey 命令测试，没有出现缺陷，请问 Monkey 测试到底怎么用？
- 我使用 Monkey 的各种参数进行测试，但想寻求 Monkey 非随机流操作测试的方法，有谁知道吗？

无论上面哪一种提问方法其实都没有说明白，例如所在的测试环境、被测对象的系统版本等。提问就和汇报缺陷一样，虽然不需要非常详尽地说明，但是还得将与问题相关的信息简单地描述出来，这样看到问题的人也比较好理解。否则问题或许能够解决，但更多的时间会浪费在沟通环节上。

不经过思考的提问一旦多了就会变成思维上的“植物人”，不能“自理”。不经过自己探索的提问对于自身没有一点帮助，也许一两次能够得到解答，解决掉手上的“难题”，

但时间一长，自己会丧失了探索问题、解决问题的能力。试问有谁会帮助你一辈子呢？学会提问的前提是学会如何靠自己去寻找问题的答案，这点会在后几节详细说明。遇到自己解决不了的问题，可以找相关经验丰富的工程师咨询，这样既能够解决问题，也能够不停地锻炼自己。

回答问题同样需要时刻牢记必须理清问题，比如，当遇见测试同行或者自己团队里的队友问："敏捷在我们团队中应该怎么实施？"或"robotium 适合做自动化吗？"这类问题的时候，我们不应该简单地回答是或不是，更多的需要先全面地思考并理解问题，然后再做解答。比如对第 1 个问题可以这样回答：

第 1 步：我们需要先阐述敏捷原本是怎样的，倡导的思想是什么。

第 2 步：在业务、团队、人员水平不同的情况下分别应该怎样实施。

第 3 步：结合一些目前的实际情况做进一步解答。

所以，当我们要思考或回答一个问题之前，有必要先了解清楚和问题相关的一些信息，就如同我们要测试一个应用一样，只有了解得足够多，才能够设计出更全、更好、更深入的用例场景，在这点上我与大家共勉。

A.2　正确地自我审视

正确地自我审视其实指的就是正确的看待自己、评价自己。也许很多人在面试过程中会被要求评价一下自己这样一个问题，虽然看似简单，但实际上很多人无法正确地评价自己，其实每个人都需要正确地自我审视。我为什么特别将这一条和测试人员扯上关系呢，原因是测试行业存在着严重的浮躁气息。测试人员往往不能正确地审视自己的原因有以下几点：

- 不知道自己要学什么。
- 自己在公司里闭门造车，不关心外界的发展。

- 自己容易受其他人的影响，这类人往往活跃在 QQ 群以及一些测试论坛上。
- 自己一直被看不起。
- 自己没有认识到“测试”是什么。
- 不清楚自己公司里测试如何发展，如何进步。

我问过很多从业人员这样两个问题。第 1 个问题是：“你为什么选择做测试，而不是做其他工作呢？”，出现率很高的答案如下：

- 我是女生，我觉得女生适合做测试。
- 我朋友做测试，我觉得不错，所以我也做测试。
- 我面试开发不行，就找测试了。
- 我觉得测试比开发容易，轻松。
- 我觉得我比较细心，有耐性。

在很多学校还没有开设软件测试这门课程之前，我认为这些回答都没错，关键是这些观点基本上或多或少受到别人的影响。建议在准备进入测试行业之前，先对此行业做个基本的了解，测试到底要做点什么？测试在整个软件工程中处于一个什么位置？等等。仅仅是因为受别人的影响就感兴趣或找工作，又怎么会长久呢？就如谈恋爱结婚，不能因为很多人说你和某位姑娘很般配、很合得来，就马上觉得自己能和这位姑娘可以过一辈子了吧？一样的道理，请真正地问一下自己为什么做测试。

第 2 个问题是：“如果有很多人和你竞争这个岗位，你为什么觉得自己比其他人更适合？”，出现率很高的答案如下：

- 我做这个行业都超过 5 年了。
- 我觉得我比其他人都要细心。

所以这两个答案真的很虚，说白了就是自己对自己完全没有信心。其实这两个问题不仅仅是初入测试行业的人不知道怎么回答，很多人在自己的行业上辛辛苦苦干了一辈

子，又如何呢？以移动互联网创业来说，我相信各位创业者都是经验丰富的前辈了，但又有几个人成功创业、开始盈利，亦或造福社会呢？再举一个例子，找工作的时候往往需要给自己定一个期望薪资，测试行业大部分人的期望薪资不是靠衡量自己能力得来的，而是“我干了多少年就该给什么价”。这两者或许有一定的因果关系，但对企业来讲真的有很大的关联性吗？答案可想而知。我建议这些从业人员不要自欺欺人了，静下心来好好地学习，充实自己，考虑一下如何将自己所在企业的测试工作做到极致。否则，现在抱怨薪资少，过十年二十年，依然还是只有抱怨的份。

A.3　不要被业界世俗的讨论蒙蔽

随着移动互联网的盛行，更多的人涌入了这个行业，很多人选择了做测试工作，原因我就不多说了。在这样一个行业中，我们需要学会判断是非，然后自己思考。在如今这样一个充斥大信息量的时代，不被信息所冲垮，而能清醒地保持对自我的正确认知和判断难能可贵。任何一个行业都有学术派和实践派，测试行业也不例外。我们经常会碰见如下的场景：

- 到处都在谈论黑盒功能测试无用论。
- 时不时地讨论测试人员是不是会灭亡。
- 今天 Github 开源了一个厉害的框架，明天 Google 又推出了新的测试技术。
- 开发测试比可以非常悬殊，比如 Google 是 10：1。
- 听说 FaceBook 根本就没有测试人员。
- 被虚名捧出来的教授一会儿说敏捷，一会儿说探索性测试。

当然还有很多场景，我在此就不一一列举了。那么这些言论到底哪些应该接受，哪些又应该过滤掉呢？这需要我们对问题有一个自己的观点，才能有基本的判断能力。接下来一一分析上面的这些言论。

到处都在谈论黑盒功能测试无用论。

首先我们需要清楚地认识到谈论黑盒功能测试无用论的人，无外乎就是那些每天只知道执行用例，甚至点点鼠标，拿不到自己期望薪资的人。有哪位听到过企业测试经理这样说的吗？如果我们认为自己不和这些人在一个层次上的话，那根本就不用在意。功能测试有没有用不在于它的形式，而在于其测试用例和场景的设计上。无论是功能、性能还是自动化测试都无法脱离用例设计。

时不时地讨论测试人员是不是会灭亡。

也许在近几年引发该争论的最大源头在陈皓（左耳朵耗子）写的一篇 QA 是否应该存在的文章。当我看完陈皓的博客，不得不表示赞同。真正认真看过这篇博客以及陈皓其他文章的话，就会知道其观点并不在于 QA 本身是否会消亡，而是那些做事没有激情，吊儿郎当的测试人员们应该消亡了。优胜劣汰不是很正常吗？现在有一部分人在讨论这个问题的时候往往连文章都没有看过就开始起哄，有必要去理会这类人么？

今天 Github 开源了一个厉害的框架，明天 Google 又推出了新的测试技术。

我们往往会在很多地方看到层出不穷的新技术、框架、方法等。我们并不是要去否认它们，更多的是要不停地接收新事物。但是应该结合自己所做的业务去接收，而不是一味地被这些新鲜事物牵着鼻子走。每家企业都有适合自己产品业务的工具、框架，但不代表这些工具就适合我们每一个人。

开发测试比可以非常悬殊，比如 Google 是 10：1。

这个观点其实很早以前业界就知道了，但是不知道为什么最近几年又被拿出来说事儿了。其实讨论这类问题没有太大的意义，其一，这只是某家企业的文化、制度，根本说明不了任何问题。其二，Google 工程师的总体素质本身就比较高，开发测试比是 10：1 不代表做测试的仅仅是那个 1。

听说 FaceBook 根本就没有测试人员。

这个问题就和上面那个问题一样，是个别企业的个别现象罢了，没有必要深究讨论。

被虚名捧出来的教授一会儿说敏捷，一会儿说探索性测试。

测试行业各个沙龙、会议非常多，新浪微博上面带“V”的“教授”也非常多。他们一会儿说 CMMI，一会儿说敏捷，一会儿又说探索性测试。很多人开始盲目崇拜，请问意义何在？崇拜前先问自己几个问题：你认识这个人吗？你和她（他）见过面吗？“教授”说出来的就一定是对的吗？说的那些对你真有实际上意义的帮助吗？对于这样的人，如果你连面都没有见过的话，请不要急着叫老师，请不要急着崇拜，作为测试人员，对任何事情我们都需要学会自己判断，跟风永远只会在浪潮末端。

测试也好，做人也罢，我们都需要让自己具备正确的判断能力，这样才能够不被多变的环境所吞没，不会迷失自己。

A.4　寻找测试的本质

我们每个人在企业中都扮演着各自的角色。我相信大家定位自己的角色不仅仅是找产品的缺陷，但是业内还是有很多测试人员没有找到测试的意义，甚至不知道自己做测试到底该做什么。

很多企业的管理层或者老板觉得测试就是保证产品质量，整天在“逼迫”产品不停地产出缺陷。无论大佬们说测试行业如何如何好，很多人在进入行业之后才发现并非如此。但是，我们也不能因此就迷失了自己，迷失了做测试的意义，不能日复一日机械地为了测试而去测试。我们在工作中需要不停地思考，我们所做的一切到底对自己有什么帮助。我们需要找到一个目标，只有有了目标，在自己遇到挫折的时候才不会轻易放弃。

很多测试人员会抱怨：每天在机械地重复黑盒测试、薪资很低或不知道如何提升自己等等。原因就在于他们并未在工作中找到自己所做事情的本质，所以就算做其他工作他们也会有同样的抱怨。我们不妨从以下几个角度对“意义”进行进一步的思考：

- 当你在执行测试用例的时候，意义在学习别人写用例的思路，学习设计方法，而不在于重复劳动上。

- 当你在编写测试用例的时候，意义在于学习怎么能够更好地分析需求，写出有意义的用例，而不在于为了完成任务，写成千上万条用例，完成所谓的绩效考评。
- 当你觉得每天的工作仅仅是不停地寻找缺陷的时候，意义在于学习研究各种方法，运用各种技术找到质量高的缺陷，并分析总结，而不在于为了去完成寻找缺陷认为指标。
- 当你作为一个测试管理者的时候，意义在于你要学习管理，你要引导测试人员学习和进步，要学会体谅和沟通，而不在于仅仅把他们当成是一个执行者。
- 当你面对一个周期很短，测试人员又很少的项目的时候，意义在于你要学会评估风险，合理使用不同的测试策略，从而积累经验，而不在于仅仅抱怨或破罐破摔。
- 当你参加沙龙交流活动，意义在于学习别人的长处，总结分享的知识点哪些是可以提炼提出来用于目前工作当中。不在于盲目崇拜，被别人牵着鼻子走。

最后，当你觉得做测试没有意义的时候，那么问一下自己为什么会选择做测试。测试工作为你带来了什么，测试工作又让你学到了什么。如果实在找不到什么值得坚持下去的理由，那么还是乘早离开这个你不喜欢的工作岗位吧。

A.5　主观能动

这点也许是测试从业人员最重要的品质——要主动（包括读本书也是主动的一种表现）。测试行业一直有很多非常“好学”、求知欲很“强”的人提出一些问题，比如：

（1）我想学习测试，请问怎么学习?

（2）我想学习代码，请问怎么学习?

（3）我所在的企业，老板一直不重视测试，我该怎么做?

（4）我想参加活动认识更多的人，学习更多的东西，我应该怎么获取消息?

粗略一看，好像能够提出这些问题的人已经不错了，至少是有学习的意识了，其实归根结底先要解决一个问题，就是自己不主动的毛病。很多人说测试行业适合性格内向的人，同时需要细心的女性 ，这点我不否认，但是这些和测试没有非常必然的联系。无论男女、性格，作为测试人员必须学会要有很强的主观能动性。也许有的人觉得自己很主动了，主动性也得方向对，否则最后只会是被动了。我们来看如下几个场景（切勿对号入座)。

（1）你是否经常主动地在 qq 群、论坛上提问或解答问题?

（2）碰见问题你是否经常主动谷歌解决方案，在官网、stackoverflow 等网站找到解决方案后尝试解决问题?

（3）你是否经常主动根据项目需求和要求日复一日地完成任务?

（4）你是否经常主动对项目需求提出质疑以及系统地学习项目涉及到的知识，从而更好的进行测试活动?

（5）你是否经常崇拜某些“教授”？

（6）你是否经常主动地参加有价值的会议以及认识有价值的人?

（7）你是否经常主动地学习你认为有价值的知识?

（8）你是否主动地定期给自己制定计划，结合公司项目以及行业的发展趋势学习知识?

如果你对以上某些奇数序号的问题回答了是，并不是说有什么不对，但我建议适当地减少这些“主动”的行为，或是在做之前了解更多的信息，这样至少保证方向不会有大错。如果对偶数序号的问题回答是的话，这是一种良性的主动方式，请继续保持。

在我进入测试行业这 3 年多的时间里，发现虽然国内测试行业发展时间尚短，且业界一直有声音说国内外测试技术差距很大，但我却认为，国内外的技术差距其实并不大，

唯一决定差距的就是人的意识，也就是主动性。不难发现，每次测试大会国外的测试从业人员并不会分享非常高深的技术，而是自己长年积累的一种方法或经验（当然，不排除他们不愿意分享非常落地的东西）。而与之相比，国内测试人员并没有养成主动总结测试方法、积累测试经验的习惯，那么时间一长自然就拉开了距离。

我是一个在移动互联网各种项目做测试的工程师，在自己刚入行的时候，为了学习新的知识，不停地加入各种讨论群。大家都知道在 Android 自动化测试框架中有一个框架叫做 Robotium，我曾经在 2011 年加入了多个与 Robotium 相关的国内的讨论组，同时也订阅了 Robotium 在 Google Group 上的一个讨论组。一个月过去了，国内的讨论组无论是提出缺陷还是关于框架实践问题的都凤毛麟角，而 Google Group 中却已经有了 300 多封讨论邮件，随着时间的推移，差距就慢慢地出现了。当然，这也和国内外教育、文化上的差异有关系，造成了国内很多测试人员普遍主观能动性较差，不善于思考的状况。但是如果你选择做了测试，并且想做好这位工作，那么就必须大大提升你的主观能动性，主动到网上查找资料，主动找人进行沟通，主动进行技术的实践等，只有这样，状况才会有所改变，否则我相信做不了多久你就会唉声载道。

本节所说的主动不仅仅是单方面的索取，还包括主动进行技术和经验的分享。也许有些大牛认为有些核心的技术、方法不能拿出来分享，这会降低自己的竞争力。但从长远来看，任何的技术或流程都会推陈出新，并且在这一行更新得非常快，当回过头来再看的时候，会发现所谓的核心技术已是过时的东西了。人生最大的魄力在于放弃，放弃自己所知所得，让自己永远处于一个原点，不停地输出自己知道的，不停地吸收自己不知道的，何乐而不为呢？每个人都是一个普通人，没有一场战斗、革命是靠一个人获胜的。一个人的能力毕竟有限，只有大家都主动将所知分享出来，才能够产生更大的价值，整个行业才会进步。

A.6 你真的会使用搜索引擎吗

移动互联网这些年无论是管理模式还是技术发展都非常快。在我当初刚做 Android

应用测试的时候，网络上的资料很少，一些从业人员碰见困难也许只能上论坛发帖求答案。经过 3 年多的发展，我依然看到一些已经有很成熟解决方法的问题被一次又一次地无脑提出。

在现在这样一个网络非常发达的时代（虽然有一些障碍，但是做 IT 的朋友怎么能不会科学上网呢？），有什么问题是网络上找不到的呢？请不要过于自信，要相信在你之前有很多的人走过和你一样的路，碰见过和你一样的问题，最终都得到了完美的解决。而这些都能够在网络上找到，关键在于如何找。为什么有的人碰见问题能够很快地定位问题，寻求解决之道。为什么有的人只会说网上资料很少，求助一些无聊的 QQ 群呢。

这的确验证了一句话"人和人之间的差距的确比人和猪之间的差距还要大"。朋友们，你们先问下自己吧，你们真的会使用搜索引擎吗?

就我个人的建议，Google 是搜索引擎的不二选择，其他搜索引擎不是广告太多就是搜索权重存在一定的利益关系，个人比较反感。使用 Google 之前，做 IT 的朋友都应该先学会如何科学地上网。



提示： 虽然我国的自贸区声称会开放访问 FaceBook 的权限，但大部分人应该没有这个福利待遇吧。

推荐一个正确科学上网的工具——goagent，这是一个在进行开发测试工作前必须先配置完的一个工具，如果有读者还不没有使用的话，具体可以上网查下使用方法。之后，我们才能够正常的使用搜索引擎 Google。

在使用 Google 的过程中我们需要注意一些关键点。很多朋友发现搜索不到自己想要的东西，或是在搜索的结果中排在后面很远才是自己想要的。要想避免这些，必须注意以下几点：

- 尽量避免整句搜索，比如一般问题都是"××为什么没有××"或"××出现了××错误"，不要直接输入一个整句进行搜索，而要学会减少搜索的字符串。比如接下来要进行 goagent 的配置了，不要打开 google 搜索"goagent 怎么使用"，这样搜索出来的结果都是垃圾结果。应该直接输入"goagent"，按下回车，这样第一页往往都是你想要的结果。

- 尽量使用多个关键字以空格间隔的方式进行搜索。大部分的问题都能够提取出多个关键字，而最好的搜索方式就是输入关键字并以空格间隔开来。比如我们有一个问题“python 如何在 json 和 dict 之间转换”，那么我们尽量搜索多个关键字，也就是“python json dict”。
- 尽量不要排斥官方文档。我们在日常工作中都会学习很多新的知识。而新的东西往往很少能有大量的中文资料以及实践博客等，于是一些朋友开始抱怨；不知道怎么配置，怎么安装。究其原因，是大多数人排斥英文官方文档所致，这样做对自己是完全没有好处的。我一直说我们做技术的，如果不想看官方文档，排斥国外的文献，那么还是趁早改行吧。
- 可以适当地添加关键字。有些时候为了更快地找到高质量的解决方案，一般我们会两种方法，一种是直接去看对这方面比较有研究的人的文章或者直接去问他；另一种就是我们搜索的时候可以适当地添加关键字。比如针对某具体问题的，stack overflow 就是个不错的选择，而某些技术的实践等个人觉得 csdn 的很多博客质量都很高。搜索的过程中添加这些关键字能够帮助我们更快地找到想要的答案。
- 部分很好的实践和答案在墙外。原因就不多说了，方法大家可以自行 Google。不要做井底之蛙。

搜索其实是门很深奥的学问，学会了怎么搜索之后，自学的能力和效率就会提升不少，同时你还会发现以前不曾发现的另外一片天空。这是每个人成长过程中的必经之路，否则你永远都会落后于时代。

A.7 每天都要学习

一名合格的从业人员应该每天都安排时间进行学习。现在吞噬我们时间的平台越来越多，新浪微博、微信、QQ 等层出不穷，当然还少不了各种游戏（说不定还有很多人被感情所困扰，浪费自己大好的学习时光）。网络上每天都有教授吹的满天飞的理论和

心灵鸡汤，这些当作娱乐消遣即可。对测试人员来讲，我更推荐的是每天都关心一下行业的发展，关心一下技术的发展并亲自去实践。一切没有实践的理论都是纸上谈兵。

我曾经在微博上给一些爱好者发过邮件，其中并非说了什么武功秘笈，而是谈了一下自己每天的学习点。当然不得不提的是，要感谢 evernote 这个应用。我每天按照时间流程会做如下一些事情：

- 了解 Zaker 热门的新闻；
- 浏览自己分组的微博，收藏感兴趣或有价值的内容；
- 将有价值和感兴趣的内容保存到 evernote；
- 在空余时间将当天保存的内容消化掉，并选择性地实践；
- 按照自己的学习计划啃一本书的部分内容。

在浏览微博的过程中，往往无法静下心集中精神来更好地去思考其中某些文章的含义以及技术细节，我们能做的就是在碎片化的时间内将我们觉得有价值的内容收藏起来。但是不要仅仅收藏起来就结束了，这和“书非借不能读也”是一个的道理。收藏起来之后我们必须抽出时间去看、去读、去了解、去实践。我平时不习惯在路上拿出电脑看文章，所以就使用 evernote 保存相关文章，这样在手机上也能够随时随地地去学习。段念曾经说他一年会阅读非常多的书籍，这点在我曾经和段念交流的过程中深有体会。他给我的引导不仅仅是技术上的，更多是思想上的。提升一个人的学识、思想境界的方法只有一个，那就是不停地学习。

在业内，很多人会嚷嚷着说要做自动化。首先要说明一点，没有银弹存在；其次可以说这些嚷嚷的人肯定都是没有做过自动化，或者用心去思考过自动化的。任何一件事情首先要去做，然后深入学习，再做评论。自动化就真的那么好做吗？自动化就真的那么万能吗？你们只听到自动化能够持续集成，能够提升测试效率，却又如何知道做自动化过程中的艰辛。万事开头难，要让自动化很好地融入项目，其开头和维护更难。

测试行业还在不停地进步、改革中。太多人在忽悠，太多的人是拿来主义。扯淡的大部分是一些脱离一线很久的人员，也许你们曾经有光辉的历史，也许你们拥有很强的

光环。那又如何？不要以为仅仅看过书，看看别人的实践就可以当作是自己的东西拿来分享，拿来忽悠从而谋取利益。没有实践的理论都是纸上谈兵。请不要没有经过自己实践就来扯自动化、敏捷、探索性测试等话题。工作要踏踏实实，学习也是一样。自己实践得来的知识才是王道，才是自己的真理。学习是一辈子的事情，对日新月异的 IT 行业更是如此。

A.8 学会判断轻重缓急

在常年累月的工作中，很多测试从业人员会碰见以下一些问题：

- 项目多、人手少，工作安排不过来；
- 不知道哪些应该做自动化，什么时候开始做自动化；
- 不知道如何很好地管理缺陷、用户反馈、客户反馈，以及各种来源的问题反馈。

测试其实是非常考验执行者策略的一项活动。多个项目并行的时候，也许项目本身对于企业来说有着不同的优先级，到了我们每个测试人员手中的时候，也应该有相应的优先级，这样才不至于手忙脚乱。因此需要学会合理安排时间进行测试，而不是被项目的安排控制住。当然也许有朋友会不服气，觉得自己项目紧急得根本不是我能想像的。我还是那句话，人是活的，重要的是在于自己的策略和沟通。

A.9 小 结

本章对做一名合格测试工程师需要的修养做了介绍，在我看来主要就是吐槽，不过我相信对大家还是会有所帮助的。



附录 B　测试行业常见问题（Q&A 篇）

测试行业发展到现在也有一段时间了。无论是测试的初学者还是已经跌打滚爬几年的从业人员都会有自己的疑惑。很多问题在各种平台上已经被问得烂掉了，作者平时也闲着蛋疼的在 QQ、沙龙、知乎等平台帮助大家，解答一些问题。当然，赞同和被抨击者各半。每个人的成长都必然会伴随着赞同和反对两种不同的声音，这再正常不过了。所以笔者觉得应该在本书中增加 1 章专门针对常见的问题进行解答，也希望测试同仁们无论是赞同或是反对，我都希望听到你们的声音，只有这样才会进步。

Q1：没有做过测试的人怎么入门？

一般问这个问题的人无非是学生或其他想转行的人。我常常在面试的时候和学生扯一些非技术问题，比如行业中重要的新闻、知名的企业等。有部分学生直接回答我说，因为学校里一直专注学习，自己没有工作经验，所以对这些不怎么关心。什么叫入门呢？进入一个行业并非一只脚跨出一步就算入门。入门的前提是发自内心地愿意去学习相关的知识，关心其发展趋势，了解更多的相关资料等。在学校或者其他岗位上的时候没有习惯去关心、学习想要从事的行业，谁又会相信这样一个人进入新岗位之后马上就会奋发图强？这样简单的道理我原本以为谁都应该懂，但面试的时候一再听到前面那样的回答，实在令人心寒。

说到入门，很多人想到的就是去培训机构学习。随着测试行业在中国发展得越来越成熟，培训机构也越来越多，阿猫阿狗为了赚钱都开始办培训机构。不得不说测试行业人傻钱多的现象已经屡见不鲜了，这才让很多人看到了商机。IT 行业的培训费用一直居高不下，那是不是应该去培训呢？培训的效果理想吗？我个人并不反对去培训，毕竟我自己在线上线下也出席各种会议进行演讲，其实也是变相的培训，只不过大都是公益免费的。如果初学者或者想充电的测试人员考虑参加培训的话，我只希望先弄清楚以下几点：

- 培训不是万能的，不要认为培训能够解决一切问题。
- 不要太关注平台，要好好看清讲师的风格和水平。
- 不要期待培训机构能够提供好的岗位。为什么？这个都需要问么？
- 行业里很多的培训课程其知识点都落后于行业发展，讲师也大部分远离一线项目。试问这类讲师会分享出很好的知识和案例吗？
- 无论参加什么培训，自身的努力必不可少。如果想不劳而获，那么做什么都是徒劳。
- 我个人不建议参加一些纯理论的培训。

说到这里，肯定有朋友要说我这是在和所有培训机构作对了。我有说过“绝对不要参加培训”这样的话吗？没有，我是希望参加培训的人能够擦亮眼睛，在选择好平台之后自身也要努力才行。另一方面我也希望培训机构能够务实，在赚钱的同时能够给学员传授更多、更有价值的知识，给行业带出更多的人才。可惜，至少目前我还没有看到培训这一业务有一个良性的发展，但是我却希望在不远的将来能够看到。

看这个问题的朋友肯定要吐槽，说我说了那么多，那入门到底应该怎么入门呢？我推荐做以下几件事情：

- 选择一个测试领域的切入点，比如做互联网测试、移动互联网测试、医疗器械软件测试、ERP 测试等。
- 去了解测试是什么，测试要做什么。推荐 3 本书。《测试之美》、《微软测试之道》、《Google 软件测试之道》，对于移动互联网从业人员来讲还有第 4 本推荐的书——本书。
- 阅读《Rework》和《高效能人士的七个习惯》这两本书两遍以上并思考。
- 了解并学习所选行业领域的专业测试知识，比如常用的工具、方法等。

我觉得以上这些就足够了，真的用心去做的话，收获可以说是一生的。不仅是测试行业，所有行业入门都是这样的套路，入门不仅仅是知识面、认知度的转变，更多的是要去选择正确的方向。

Q2：测试工程师要具备什么能力？

如果你认真地从头开始阅读本书之后（尤其是附录 A）还提出这样的问题，那么问题在我。如果没有，那么请重新认真地阅读本书。如果这个问题有上下文，那么的确是一个很好的问题，但请千万不要这样单纯直白的问出来，否则被问的这个人肯定会惊惶失措的，小伙伴们也会惊讶的。

Q3：测试比开发技术含量低吗？

我们先来看一下提出这个问题的测试人员自己是怎样看自己的，开发在他们眼中又是怎样的呢？

应聘岗位的时候

眼中的自己：觉得根本就没有提任何的技术问题，面试官根本就是在和自己扯淡。

眼中的开发：根本就是火星人在互相交流啊，不明就理。

编写测试用例的时候

眼中的自己：觉得根本就是没有技术含量的重复劳动，默默的耕耘，还不被重视。

眼中的开发：项目的顶梁柱，底气十足。

发薪水的时候

眼中的自己：月光族。

眼中的开发：高富帅。

向别人自我介绍的时候

眼中的自己：毫无底气，介绍的时候都不敢喘粗气。

眼中的开发：他们都是外星人。

和开发争论问题的时候

眼中的自己：普通人类，战斗力 1。

眼中的开发：超级赛亚人和奥特曼的合体。

写到这里我自己都忍不住要笑了。先不说谁比谁技术高，自己都看不起自己的人，根本就不要指望别人会看得起你。在讨论这个问题之前，我们首先需要抛弃上面这些非正常人类的思想。

《灌篮高手》相信很多人都看过吧。足球篮球这类运动靠的都是团队合作，每个人都有自己的职责，并非说 11 个人全部是前锋或者守门员，就一定会赢或会输。测试和开发人员也是一样，都仅仅是项目团队中的成员，每个人都需要发挥自己最大的能量，才能够很好地完成一个项目。所以这两者根本就不在一个纬度上，有啥可比的呢。

很多朋友说自己做测试天天都在重复劳动，天天都在手动工作，觉得开发写代码非常的神秘，薪水很高。我们在贬低别人或者羡慕别人之前先从自身找原因。很多测试从业人员刚开始都是从学习测试、编写测试用例开始的，任何一家公司也不是开始就有很完善的自动化测试，一切的一切都是需要经过一个艰辛的过程。也许你在做黑盒测试，也许天天在做重复劳动，但没有人剥夺你学习的权力，你可以自己学习，将手上的工作和测试活动越做越深，那对自己对企业都是很有价值的事情。

再来说开发人员，写代码是开发人员最基础的要求，但不是会写代码就代表是一名好的开发人员。有相当一部分开发人员每天也是到处复制拷贝的代码，修改小缺陷，其实同样也是重复劳动，他们也会怨声载道，只不过不如测试人员抱怨得那么频繁，影响力那么大。

家家都有本难念的经，就是这个道理。任何一个岗位、任何一个人都有自己的苦恼，没有什么可比的。这类无聊的问题和想法没有必要花时间去思考，要我说有时间思考这些问题还不如多学习点知识，看点书，总结点经验来得实际。

Q4：参加测试培训能有多少提升？

所有参加培训的人几乎都会问这样几个问题，“培训好之后能拿到多少薪资呢？”、“培训好之后能找到比现在好的工作吗？”、“培训好之后能学到点什么呢？”。培训不是银弹，现在行业的培训无论务实务虚，最终还是起到了一个辅导、指引方向的作用。俗话说的好：“人与人的差距并不在工作和睡觉的那 16 个小时，而是剩下的 8 个小时”。无论参加何种的培训或是在何种的企业团队，自己的总结和实践是不可或缺的，仅仅靠聆听无法真正掌握一门技术和知识。

Q5：黑盒测试有价值吗？

肯定有。就拿移动互联网的测试来讲，黑盒测试的价值是任何工具或者自动化测试无法替代的。移动互联网的应用与其他软件相比主要有两点不同：其一是从某种程度上来说用户体验比任何功能都重要；其二是更新速度奇快。没有一种自动化能够判断用户体验的好坏，同时在快速迭代中，不合理的自动化只会给技术人员带来不必要的工作量和压力。

如果黑盒测试的价值毋庸置疑的话，接下来也许有朋友要问，黑盒测试的价值体现在哪里呢？黑盒测试很多人都在做，为什么感觉看不到价值呢？这主要有两大原因：

原因 1：大环境所导致。大部分的管理者还是看重技术，认为技术好的就一切都好，却不去真正地关心技术是否适用以及怎么使用。自然而然，很多人就觉得黑盒没有价值，只有自动化才是唯一的出路。

原因 2：很多测试人员根本不了解黑盒测试的价值，也根本找不到黑盒测试的价值，从内心也不愿意去深入了解黑盒测试到底做些什么。答案不是唯一的，在不同领域，黑盒测试关心的点也会截然不同，但测试人员不愿意深入理解黑盒测试却是一个普遍的现象。原因是什么？或许还是因为原因 1。

很多黑盒测试工程师就知道去背“等价类”、“边界值”，我请问你们平时真的会用到这些方法么？有人会回答当然用到了。是的，你是能够很自豪地正面回答，但这难道就是黑盒测试真正的价值了吗？这些都是测试最表层的东西，比这更重要的是提升测试人员自身的内涵。

移动互联网应用的黑盒测试无非就是两大点，如你能够熟练掌握这两大测试点，那我可以说你比任何一个测试人员或是其他岗位的人都来得有价值。

1. 功能场景测试

功能测试大家都是知道的，应用本身都有功能模块，功能场景测试的范围其实非常广。应用的最终使用者还是广大用户，他们使用各种式样的移动设备、软件系统，他们

会在地铁、厕所甚至飞机上使用你的应用，他们以各种你想不到的方式来“折磨”你的应用……这些都是所谓的功能场景测试，也就是模拟应用在真正的用户、真实环境下的使用。要做到这点容易不容易我就不多说了，请大家问自己一个问题：“你真的知道自己产品的受众是谁吗？”当然，请不要回答是全球的用户或者正在使用你应用的人。乔布斯敢不敢说这句话我不清楚，但如果你也可以，那么你可以醒醒了。

2. 视觉、交互测试

在本书的第 3 章详细地描述了什么是用户体验测试，从中可以了解到，要去做所谓的用户体验测试并非容易。在进行用户体验之前，首先要测试应用的风格是否统一、交互是否合理，这些并不是靠一个测试人员苦思冥想或者长年累月的测试经验就能够获取的，更多的是需要有针对性地进行学习和总结，才能够慢慢地有所提升。

Q6：手动测试有价值吗？

手动测试当然有价值，但是纯粹的手动测试就毫无价值了。在开发周期中，每个人都会去使用自己的应用，这个过程其实都是在手动测试。我们要有明确的目的性和测试切入点来手动测试，将手动测试的效率提升到最大才是有价值的。在很多人看来移动互联网应用的手动测试就是点点手机，玩玩应用。的确，哪一行说到底都是在手动工作，关键就在于你怎么玩得转了。

Q7：怎么做移动互联网应用的自动化测试？

首先我想说的是，这类提问的方式非常的没有营养，希望大家以后问问题要有技术含量。没有任何前提，没有任何切入点直接就抛出这样一个问题知道如何回答。现在网络那么发达，大部分技术都是可以通过网络入门的，那么提出来的问题应该是比较具体的、更具实践性的问题了，而不是像这样让人觉得摸不着头脑。

言归正传，问题还是要回答的，哪怕别人问的问题什么上下文都没有，我们也应该很专业地来回答这个问题。我个人建议从以下几个方面考虑应用的自动化。

界面自动化——Robotium、Appium

这两个框架在之前的章节中都已经介绍过，这里就不细说了。Robotium 的发展历史比 Appium 要长许多，版本更新也很迅速。



提示： 前不久我的朋友李正还（新浪微博：@喜---力）为了让 Robotium 更好地为国内服务而创办了 Robotium 中文网（www.robotium.cn），在解决测试同仁对 Robotium 使用中的疑惑的同时，也在努力地翻译 robotium API 文档。

Robotium 是一个使用 Java 语言在 Android 的 Instrumentation 框架的基础上，封装了模拟用户操作的接口的测试框架。工程师们可以使用 Robotium 对 Android 应用做界面进行自动化测试、接口测试甚至单元测试，Robotium 的开源也使得我们更方便扩展其功能从而为自己的产品提供更好的自动化测试服务。目前最新版本也更好地支持，有 webview 控件的应用，在其发展历史中受到广大 Android 测试工程师的青睐。

再来说说 Appium。Appium 可以说是移动互联网应用自动化测试的后起之秀，它的出现的确让我感到狂喜。Appium 的特性吸引了无数人的眼球——同时支持多种编程语言、跨 Android 和 iOS 平台、很好地支持了混合式应用（Hybird Application）等。Appium 在 Android 的自动化测试框架中，引用了只支持 Android4.0 以上的 uiautomator，而在 iOS 上引用了 Apple 的 UIAutomation 库，从而实现了元素定位和操作。



提示： 就在 Appium 出现的同时，TesterHome 也随之出现（www.testerhome.com），TesterHome 是一个由国内一些测试爱好者自发成立的团队组织，他们在对 Appium 做了翻译的同时，也为国内很多从业人员答疑解惑，在这里要感谢两位同学的付出（新浪微博：@seveniruby 和@晋恒温）

分层测试

在移动互联网的应用测试中，我提倡进行更多地“分层测试”。随着时间的推移，

越来越多的沙龙中也频繁的出现了这样的概念。分层测试的主要理念是“从前到后，从上到下”。对移动互联网的应用来讲更是如此，针对每一层用不同的编程语言、不同的测试工具进行测试。这里要强调一点的是，分层测试是一种思想，并非一定要自动化。我这里举个 Android 的例子：

- UI 层入手：Monkey、MonkeyRunner、NativeDriver、Sikuli、Instrumentation、Athrun、Cafe、Robotium、Appium 等。
- 应用代码层入手：robolectric、Instrumentation、TraceView、Java、C 等。
- 前后端结构入手：Instrumentation、Java、Python 等。
- 网络相关入手：Fiddler2、tcpdump、Charles 等。

其实现在很多企业的应用依然没有去做分层测试，而是将所有的测试活动压在手工集成测试上面。测试人员和开发人员压力大的同时，也无法搞清楚缺陷到底发生在哪一层，这样往往会浪费很多的时间。所以在测试活动中，分层的意识是必须的。

自动化提升黑盒测试的效率

测试并非只有全手动和全自动化测试两种，实际上，使用这两种极端测试法的团队或者项目也是极少的。某种程度上，对移动端的大部分测试还是处于黑盒测试的较多，而自动化或者说脚本只是为了提升这些黑盒测试的效率，从而达到质变的目的。我在以前的沙龙中提到过，勿以自动化小而不为之。只要我们仔细地审视一下自己的工作，会发现有很多测试点的效率都可以被提升，关键在于你愿不愿意去做。

Q8：测试人员选择进入大公司还是小公司？

就我个人而言，对于移动互联网的小公司，没人比我更有发言权的了。我在移动互联网经历过 3 家创业企业，并且负责将测试组从零开始带起来。对于这个问题我并不认为可以直接回答，因为前提还能够分得更细，具体如下：

进入创业企业

创业企业的团队人数少，节奏快，流程大多很随意，同时每个人承担的事情也会更多（简称多面手）。就测试工程而言，我个人觉得，如果没有一个很务实、技术很好（记住，不是理论派的那种人）的测试领导的话，那么就根本不要考虑创业企业了，进去了只会越来越迷茫，闭门造车罢了。在创业公司工作的同时，一定要记得不定期地和外界其他企业的测试同仁互相交流，提升自己的认知水平，否则随着时间的推移，自己会慢慢地失去竞争力。

进入知名大企业

大企业和创业公司的情况正相反，人数多、流程规范而缓慢、每个人承担的业务更集中，但没有创业公司那么自由。俗话说的好："小公司学技术，大公司学做人"。大企业就像一个小社会，林子大了什么鸟都有，帮派斗争、同事关系等。但往往在大企业中更容易变成井底之蛙，与外界的沟通会显得尤其匮乏。我们需要注意的是和同行进行沟通、学习、思考，每一样都是不可或缺的。

是不是这样区分以后再回答这个问题就可以结束了呢？当然不是，在移动互联网中还有一种特别很常见的场景——大企业中的小团队。在这类团队中，开发测试比和创业公司一样很悬殊，但又没有创业团队那么灵活，导致工作很拖沓，效率很低。想要从事移动互联网工作的同学们需要特别注意这一点。

Q9：中国高校有软件测试专业吗？

我在尝试回答一个问题之前都会先亲身经历过，否则就是扯淡了。首先我来回答这个问题，中国高校是有软件测试专业的，但目前也只是刚起步。北航就是个不错的例子，貌似珠江也有了测试专业。

那么再来看一个衍生出来的问题，高校的教师是不是都理解软件测试是什么呢？答案大家看我曾经去考察过的几个学校吧。我曾经在思考为什么中国的很多测试工程师自

已连测试是什么都不知道，原因归根结底在于工作之前对测试活动毫无了解。所以我就在 2013 年年初决定前往各个高校，对学生做宣讲，为了让更多的人了解测试是什么。以下是我和几个学校具体的交涉结果：

同济大学：

几所学校中唯一一所没有成功完成宣讲的。我很诚恳地提出申请，但最终却不了了之。如果该校负责的老师看到本书的话，希望给我一个机会，能够前往同济和学校的同学们分享测试行业的形势及技术心得。

复旦大学：

首先说明我去的是复旦大学的张江校区。学生会的同学很热情，但最终结果却不尽如人意，最终在约定的时间、地点，除了学生会的同学以外无一人参加，说心里话对我打击还是很大的。

华东师范大学：

本书会和大家见面正因为我前往过华东师范大学。虽然并未举办宣讲，但和孙海英老师的交流却意外地愉快。在此感谢孙老师的热情招待，并最终鼓励我促使了这本书的诞生，谢谢。

上海大学：

上海大学是 4 所高校中宣讲最为成功的。不过其中依然存在些许“笑话”。上大是我展开高校宣讲的第 1 步，和负责老师交涉的时候，我说明了自己想来和同学们介绍软件测试的概念以及在行业一线具体要做点什么，让同学在真正进入行业前对此有一个了解。那位老师略有自信地和我说，我们学校的同学不需要宣讲，学校本来就有软件测试的课程。顿时我就惊讶了，心想我毕业也就几年的时间，高校进步真快。但经过接下来的交流之后我才发现，这位老师所谓的“软件测试”课程其实就是“软件工程”。同时这位老师坦露到计算机学得好的学生出去都做开发了，薪水也都不错，就算做测试肯定也是一等一的。我心中只有无限的叹气。言归正传，上海大学之所以宣讲能够非常成功，在于上海大学开源社区的接口人杨瑒，他是透露着 Geek 气息的年轻人，

在他身上我看到了中国软件将来无限的希望。

所以就通过我自己的实践证明了以下几点：

- 很少有学校有软件测试专业。
- 学校中的老师对软件测试根本就没有概念，误认为是软件工程的数不胜数，“学好编程走遍软件天下都不怕”的理念不知会误了多少测试人的前途。
- 学校中的学生对于软件测试大多不感兴趣。一方面是因为对于软件测试一无所知，另一方面就是老师的误导。部分老师传授同学的思想就是开发比测试薪资多，所以要学开发。
- 学校相对还是不够开放。无论是学校还是老师对我的任度都不高，不止一次地，在我主动做自我介绍和此行目的之后，依然对我抱有很大的怀疑态度。

通过以上这些事实，我们不难得出目前中国测试行业的从业人员为何是这样一种状态的原因，也证明了我自己的想法：解决问题要从根源开始。但既使问题解决了我还是会担忧，谁会去上软件测试的课程呢？扯淡的教授还是居多，那么课程开设的意义何在呢？不过这些都是后话了，我希望中国的高校教育能够迈出有意义的第一步—设立“软件测试”这样一门独立的学科，让真正有经验的、一线的、工程师前往交流。

对了，我还遗漏了一家高校—工程技术大学。这所学校让我纳闷的就是接口同学不止一次地向我反映，学校有规定，如我要前往宣讲，需要先付几千元人民币，至今我都觉得很不解。

Q10：小　结

Q&A 到此就结束了，我知道很多同学会觉得这些问题太过基础，但正因为基础所以才显得更加重要，也是驱动我动笔的动力。如果大家有实际测试的问题，可以直接找我咨询，我会一一做回复的。



附录 C　博客摘录

附录 C 是笔者从自己的博客中摘录了一些相对有价值的文章，希望对大家有所帮助。

编者注：附录 C 内容中包含大量专业词汇与网络词汇，为保证博文原汁原味，编辑过程中尽量予以保留，另外，博文中的一些主张与观点，仅代表作者个人思想，请读者自行吸收理解。

C.1 我们需要专职的 QA 吗？

——写于 2012-04-16 11:34:15

原文链接：http://blog.sina.com.cn/s/blog_7022adbf0100zgqo.html

左耳朵耗子的文章《我们不需要专职的 QA 吗？》，还是引起了各种风波。就我个人来讲，我非常赞同他的抱怨。注意是抱怨，并非是观点。就观点来讲，其实真的是婆说婆有理，公说公有理。就目前 IT 行业来讲，还真说不清楚。就他的抱怨来讲，就国内来讲。是的，的确，很多公司都存在他说的这样的情况，甚至更甚。我自己从一家小公司做起，自己深有体会。在这里，我可以很负责地说，如果之前的一家公司不是因为我的存在，那么左耳朵耗子几乎说的所有现象都会存在。

不过这里我要说一点的是，测试们，不要说别人看不起测试。我曾经在 Weibo 上面就写到过，中国的测试之所以被别人看不起，根本就是因为测试自己在看不起自己。

由于文章太长，我就不逐个进行评论自己的看法了。

> 我有一次私自 review 他们的 test case 的时候，发现很多的 test case 这样写到——“Expected Result: Make sure every thing is fine”，WTF，什么叫“Every thing is fine”？！而在 test case design 的时候，没有说明 test environment/configuration 是什么？没有说明 test data 在哪里？Test Case、Test Data、Test Configuration 都没有版本控制，还有很多 Test Case 设计得非常冗余（多个 Test Case 只测试了一个功能），不懂得分析 Function Point 就做 Test Design。另外，我不知道他们为什么那么热衷于设计一堆各式各样的 Negative Test Case，而有很多 Positive 的 Test Case 没有覆盖到。为什么呢，因为他们不知道开发和设计的细节，所以没有办法设计出 Effective 的 Test Case，只能从需求和表面上做黑盒。

我：前半段，我不想评论什么。我只能说写这种 test case 的 tester 太过肤浅，用我话来讲，根本就还不如那些实习了一个月的 tester。当然，在我以前工作过的公司也发生过这类情况。不过这类情况一般都是发生在第一次做测试的人的身上。至于左耳朵公

司怎么会出现这样的人，我个人比较匪夷所思。关于后半段，其实作为一个测试，几个case 测试一个 point 这个在现实生活中没有办法避免的。不知道开发和设计细节，我只能认为是一种沟通上的不当，测试应当在设计的时候就参与到项目中，并且很好去design case。就如左耳朵说的，并非只是单纯地看着文档，跑着黑盒。不过这里有一点我必须要说的是，测试真正需要的是一个有好的想法，好的 case 的设计思路，以及很好地用户体验的这样一个人。因为在我看来，黑盒和白盒只不过是方式不同，其主要还在于 case 本身。

开发人员做测试

我：关于这点，我作为一个测试，不可否认。是的，在很多场景或者很多需求上面开发人员做测试的确相比测试人员更加有效。国内的测试普遍代码基础较差，有很多几乎就没有代码基础。对于这样的局面，比如我们写某个 api 的功能测试，又或者写一个查看对象生命周期的测试，这类我敢打赌来讲，就一个项目的 dev 来讲肯定比该项目的QA 写来得心应手。

但是仅仅限于一些基本功能的保证。dev 无法保证一个系统性的全面测试，就如我上一段说了，我不 care 是 dev 或者 QA 去做测试，而是设计 case 这样一个人是不是真的想法全面，是不是真的设计得好，是否真的会从一个用户的角度去设计。左耳朵所说的，QA 要懂代码，这点我不得不说，我很赞同。并非要让 QA 去品尝自己做出来的产品失败或者有 bug 是什么滋味。而是只有懂得了，才能够真正地从逻辑上面去分析测试路径，分析测试方向。这样才不会让别人感觉到只是片面的黑盒测试罢了。我始终不明白，为什么不做开发的 QA 会比 Dev 在测试上更专业？ 这一点都说不通啊。

我：就刚刚进入测试行业的人来讲，肯定会反驳。但是左耳朵说的也并不完全对，不会做开发的 QA 只不过不能成为优秀的 QA，但是和做测试的 Dev 没有可比性其实。

减少沟通、扯皮、和推诿

我：其实就这点，所有公司任何一个测试 team 和 dev team 都有出现左耳朵说的现象。怎么说呢，其实 dev 本身不是只为了 coding，QA 本身不是只为了测试求测试。如大家真心都是为了把产品做好，各种扯皮、抱怨都会减少。其实基本上来说，国内很多

测试和 dev 的素质（不是说做人的素质）就职业的素养来讲还是不够高。这个是一方面，另外一方面来讲，好的流程一种好的沟通方式也可以大大减少左耳朵说的这类问题。原本我是想建议说可能以后可以有一种职位就是开发和测试并存的。但是后来想想，不对。我还是不赞同两者混淆起来。两者除了思维相差太大之外，其花精力以及方向都是不一样的。不可能让同一个人或者同一个 team 去完成。

吃自己的狗食

我：这点我无条件地完全同意，并且非常同意。

> 关于 SDET。全称是 Software Development Engineer on Test。像微软、Google、Amazon 都有这样的职位。但我不知道这样的职位在微软和 Google 的比例是多少，在 Amazon 是非常少的。那么像这样的懂开发的专职测试可以有吗？我的答案是可以有！但是，我在想，如果一个人懂开发，为什么只让其专职做测试呢？这样的程序员分工合理吗？把程序员分成两等公民有意义吗？试问有多少懂开发的程序员愿意只做测试开发呢？所以，SDET 在实际的操作中，更多的还是对开发不熟的测试人员。还是哪句话，不懂开发的人是做不好测试的。

我：我以前认识 MS 里面的 SDET。比例在微软里面还是蛮多的。我同意不懂开发的人成为不了真正意义上面优秀的测试。但是并非一定要让懂开发的测试去做开发。或者说两者兼做。

我不说什么“开发是创造、测试是破坏”这种理论大话了。我来说实际的，实际上面，测试是一种全面性的工作，虽然任何一个测试不可能让产品没有 bug。但是也需要无限接近于这个目标。一个人的精力是有限的，我们需要测试是因为一个真正意义上面优秀的测试人员肯定比一个真正意义上面优秀的开发人员想得更多（并非更全）。

我一直认为测试的思维是第一，其余所有的只不过是测试实施的手段不同罢了。但是就测试而言，黑盒测试用例的设计，白盒测试框架的搭建、架构，api 的封装。包括各种压力测试、回归测试、性能测试、用户体验测试，更甚的有单元测试、coverage 测试等等。这一系列要在一个项目中做好并非是容易的事情。往往测试会用各种不同的方法去达到一个目的，这个就如同测试会有 N 条测试用例去覆盖一个功能点一样。这

个从 dev 或者旁人的角度来看的确可能是一件荒唐的事情，但是测试往往是一种摸索，是一种创新，是一种比较的过程。这样在日积月累之后，case 会被精简，方法会被改进，效率会被改进，bug 也会相应的减少，上线之后的问题也会逐步减少。当然一个真正好的测试应该告诉 dev，某些类应该怎么写，某些方法应该怎么写，某些逻辑应该怎么做保护。dev 和 QA 相辅相成才能够各取所需，一起提高，不是么?

以上，所有的观点都是建立在一个有真正测试 sense 的 QA，一个真正知道要做什么的 QA 的基础上面。其实目前很多的局面的造成就是因为测试自已不知道要做什么，对于产品、对于测试都没有激情所造成的。这类测试根本就不配做测试。说实话，长此以往，第二个抱怨的、第三个抱怨的都会跳出来。到时候就如同狼来的故事一样，谁也不会相信测试。

在此，我向陈皓表示敬意。

C.2 学习让测试更精彩，测试让生命更精彩

——写于 2012-08-10 02:30:38

其实，这篇文章的名字我想了很久，最终定了这样一个标题。很多人会觉得太大了吧，学习和测试有关系，测试和生命的关联貌似没有那么大吧。其实不然，这篇文章提到的一种态度，是一种思想、是一种精神。我认为是测试同仁们互相宣传的一种精神。

为什么我想写这样一篇文章呢。

原因一：IT 行业发展历史就不久，测试这个行业的历史就更加不用去说了。在国内最近慢慢引进 Scrum 和 ET 的现在，不得不说对于 Tester 的要求越来越高，不过就如我和微软的 Bill 所交流的，并不是对于 Tester 的要求越来越高，而是国内的大多数的 Tester 没有达到一个应该的水平，所以当一种新的模型新的方式出现的时候就觉得力不从心。

原因二：测试行业正在飞速发展中，越来越多的人投入到了测试行业中。但是大部

分对于测试的理解有偏差。

原因三：同样的，由于移动互联网的崛起，目前相关的企业越来越多。职位需求量也会相对应地增加，由于对于每个产品 release 的质量要求并非非常强，导致很多测试不“敏捷”也得“敏捷”，从而迷失方向。

我来说另外一个现象吧，现象是什么呢？

面试官 A 问“你以前仅仅是做手动测试的么？”面试者大多心理会有波动，觉得瞬间低人三等。

面试官 B 问“你也做了 3 年测试了，你接下来打算做什么呢？”“我觉得我也做了蛮长时间了，接下来就转管理吧”。

面试官 C 问“你为什么选择做测试呢？”“我觉得做测试轻松。”“我原本想做开发的，但是落榜了，所以想……”

那么我想来谈谈我对于做一个测试的看法：学习什么能够让测试更加精彩呢？

一、知己识人

所谓知己就是清楚地认识自己，什么才是对自己最重要的。就测试这个职业来讲，我认为自己得到什么，学到什么才是最重要的。很多人看到这里可能觉得是正确，这种大道理谁都知道。但是平时呢大部分往往，嗯，保证产品质量，保证公司企业的质量。但是有多少测试做的事情是真正自己想做的，又有多少做的事情是对自己有意义的。可能工作本身带来不了很多的学习点或者兴趣点，但是我们不能被忙碌的工作、频繁的项目、坑爹的老板所迷惑，因为我们是测试，我们是一个需要提升自我修养，提升自我知识面才能够更上一层境界的职业。所以笔者自己是时不时地会问自己到底学到了什么，自己需要的是什么。

所谓识人，这里所说的识人不是说怎么识别好人坏人，而是如何去面试一个测试，如何给一个测试去定一个要求。为什么笔者会提到这点，就如上面所说的，现在很多人进入了测试的圈子。笔者自身是一个做移动互联网的测试，同样也经历过了上海，北京，杭州等地在各个不同阶段的面试。感觉到了不仅仅应聘测试的 IT 们迷茫，企业本身对

于测试的定位也很迷茫。面试就是第一个能够看出来的地方。

个人觉得测试这个职业很奇特，因为除了学历、技术还和这个人的各个方面素质有着紧要的关系。当然这里我不想多的举例子，我只想给各个面试官以及企业一点建议，筛选海量的简历的确可以靠曾经的工作经验，可以靠学历，但是希望在面试过程中能够从“态度”“开拓性思维”“为什么要做测试”三方面去做检查，如果发现有欠缺能够在入职之后进行相应的培训补足，这样的话，我相信对于广大测试人和企业来讲都是会看到好处。同样的会加速推动测试行业的发展。

二、找到测试的意义

这里其实就和知己很像，我相信这次 chinatest 的讲师也好，我碰见的各位同仁也好，每个人在企业中都分别扮演着自己的角色。我相信我们大家的角色绝对不会只是定位在找 bug。但是我也同样的看到很多测试人没有找到测试的意义，很多上层或者老板觉得测试就是为了保证质量，呸！他们只会觉得测试是为了找到 bug 的，无论嘴上说的多好听，很多人最后还是会用数据来定你的 KPI。但是，我们不能因为如此迷茫了自己，迷失了做测试的意义，不能最终为了测试而去测试。测试的意义在于从各个角度，各个维度去保证产品的质量。这句话是废话，也是空话。

但是为什么我想这里提醒大家找到测试的意义呢，是因为只有测试人找到了测试的意义（可能是提升自己的管理能力，提升自身的技术能力，分析能力等），那么才不会在各种困难，各种挫折面前迷失了自己，才不会为了测试而测试，最终得不偿失。

当你在执行测试用例的时候，意义在学习别人写用例的思路、学习设计方法，不在重复劳动上面。

当你在编写测试用例的时候，意义在于怎么能够更好地分析需求、写出有意义的有限的用例，不在为了完成任务，写上成千上万条用例。

当你面对找缺陷这个常见的任务的时候，意义在于学习研究各种方法、各种技术找到质量高的缺陷，分析总结，不在为了去完成缺陷数量而去找。

当你作为一个测试管理者的时候，意义在于你要学习管理，你要引导测试人，你要

体谅沟通。不在写好用例之后简单地让他们去执行。

当你面对一个周期很短、测试又很少的项目的时候，意义在于你要学会评估风险，合理使用好各种方法应对，从而积累，不在用自己的生命换取产品所谓的质量。

当你觉得做测试没有意义的时候，意义在于测试为你带来了什么，测试让你学到了什么，不在你是不是想跳槽或者转行。

三、心理素质

笔者为什么将这条放在那么前面呢，这里不得不提到，笔者在仅仅只有两年工作测试经验的时候就已经亲身经历过了身边的测试由于心理问题而最终选择绝路的事情。能从心理上真正了解测试的只有测试，这点我深信不疑。任何一个测试最先面对的心理压力就是重复性的劳动。测试人是愿意去做？是否愿意去寻求这重复劳动中的真谛？这其实是任何一个测试都应该迈过的一个坎儿。

而在之后的测试生涯中，依然会碰见很多心理的考验，自己对于质量心里没有底、或者由于产品发布问题遭到了老板的指责、或者和开发以及其他人闹不开心、或者找不到缺陷时期的郁闷、达到了测试瓶颈时候的困惑等。

测试也是人，每个人都有自己的背景以及性格，这些时间一长，往往对于测试来讲，就是考验心理素质的时候，你是否还看得清自己的路，是否还知道自己做测试的初衷，会不会对于自己做测试去质疑等等。

测试这个职业无非是心理活动波动最大的，心理上的暗示和缓解对于测试是最大的一个帮助。

笔者第一本读的有关心理学的书籍是《梦的解析》，之后陆续看了佛洛依德的若干部著作。对于心理学上很有兴趣，强烈推荐各位测试同仁有空读一两本心理学有关的书籍，相信你得到的帮助绝对不只是心理上的。

四、主观能动

很多人说测试行业中很多都是性格内向的人，很多需要细心的女性。这点我不否认，但是只是和测试本身没有非常直接的关系。但是无论男女，无论性格，作为测试必须要

学会的是主观能动。笔者在本文一开始就提到测试行业原本历史就短，并且国内外的文化、技术差距很大。我自己是一个做手机移动端的测试（如果有人要交流相关技术，我很乐意一起讨论），在移动互联网的测试，国内的积累更加的少。

我举个实际的例子，在安卓的自动化测试框架中有一个框架叫做 Robotium，我无意识中地加了国内很多讨论群，同时也订阅了 Robotium gmail 的一个讨论组。一个月过去了，国内的群很多都沉默，但是那个 gamil 的组却已经有了七百多封的讨论邮件。这里其实总结来讲，国内外的教育，文化从我们小时候开始与国外就是不同的一个理念，造成了国内很多人的主观能动性相对来讲比较差。但如果你选择了测试，那么必须大大提升你的主观能动性。如果你想做好测试，得到更多的信息，得到更多的技术，那么你必须主动去网上查找资料，主动地找人进行沟通，主动地进行实践，那么一切才会有改变。否则我相信做不了多久就会怨声载道。

同时，这里的主动 不单单是单方面的吸收，还有主动进行分享。每个人都是普通人，没有一场战斗，革命是靠一个获胜的。一个人的能力有限，当大家把自己所知的东西都主动分享出来，那么才能够产生更大的财富。一切才能够进步。

五、乐观精神（阿 Q 精神）

首先澄清一点，笔者在除了测试以外的方面并非一个乐观的人，所以还修炼不到火候。乐观对于测试绝对不可少。你往往面临着一个复杂的功能性产品，往往会被误解，往往会被很多人在心里看不起，会因为找不到缺陷而心情不好等等，等等。乐观会让你精神拥有强壮的体魄和内心，否则你会无法继续在这条道路上走下去。可能最后打败你的是你自己，说服你的是你自己。这份精神难能可贵，当你面对各种各样的突发事件，面对各种困难的时候，不妨乐观一下，调整好心态，去在能力范围内做好，会有意想不到的收获。

六、沟通能力

说到这里，如果你已经具备了测试的最基本的素质的时候，那么你绝对、绝对会觉得测试绝对不是测试唯一的工作。在一个公司，项目中测试不是你一个人的战斗。最先的一点，避无可避，也是历史上战斗最悠久的一个对手：开发。可能再好的朋友也会和

你争论得面红耳赤。

当你要确认缺陷的时候，你可能会遭到各方面的质疑；当你明确需求的时候，你可能需要和你的项目产品经理甚至客户进行沟通；当你要管理团队或改进测试流程的时候，那么你可能需要和相关的所有人进行沟通协调。

沟通是一门技术，这句话放在测试身上再好不过了。我们往往扮演着各种各样的角色，曾经有人甚至告诉我，我除了做测试，还做全职的售前售后。很多测试在为提升效率而烦恼，当你解决了沟通问题的时候，那么效率上升的比例可能是几何倍数增长的。同时，你的人际关系也会越来越好，这样会让你做管理、做协调，甚至做结构上的改变变得那么轻而易举。

沟通能力其中比较重要的就是描述。当一个测试人员描述一个事情都描述不清楚的时候，绝对不是一个好的测试人员。测试人天生需要汇报提交缺陷，而清楚地描述这些缺陷如何发现、现象怎么样是一项基础技能。描述问题另外一面就是倾听问题。用怎么样的心态描述问题，又用怎么样的态度去倾听别人所说的。决定了沟通最后的效果。

七、分析能力

我们慢慢地从一些软性条件上说到了硬性的条件上了。好的分析能力带给测试的会是另外一片天地。分析能力其中包括了：如何去发现问题，如何去分析问题，如何去解决问题，如何去总结问题。这里的问题不是指测试中的缺陷。可能是一种模型的运用，可能是一种测试技术，也可能是一种人际关系等等。

曾经在 Google 全球 code jam 竞赛中获取第一的中国选手告知我“万事不懂问 Google”。同样地我相信，很多人会觉得为什么有的问题我就查不到，别人就查得到。如何灵活运用搜索引擎真的是一门学问。好的分析能够让你找到问题出在什么地方，然后找到切入点进行相对应的改进以及修改。面对产品，能知道风险最多的地方在哪里；面对技术，能够搜寻出最终的可行性方案；面对团队，能够对症下药，而不会无从下手。分析来说，实在有太多地方可以说，我这里就不一一说明了。

八、条理性

任何事情都有轻重缓急，在《高效人士 7 个习惯》以及 ChinaTest 中柴阿峰提到的

基于风险的测试中都提到了这点。作为测试，很可能你会有很多事情排着队。可能是烦人的客户、可能是不停在变的需求、可能是新测试技术的探索、可能是自己私人的事情等等。当项目时间、测试人员数量、产品风险、个人私事这样几个维度一起向你攻击的时候，那么你只有通过分析，然后有条理地归类到 7 个习惯中提到的四象限中。对于测试、缺陷有优先级，工作有优先级，杂事有优先级，什么都要有优先级。包括朱少民老师提到的传统脚本测试和目前正热的探索性、敏捷测试的和谐并存。这也是需要有条理性地针对公司，项目的情况具体安排。并非传统不好，并非敏捷探索就一定好。不管黑猫白猫，抓到老鼠的就是好猫，不是吗?

九、责任

这点毋庸置疑。测试必须要有责任感。当然不是说让测试承担一切的责任，而是对于自己所做的一切进行负责，对自己负责。测试是一个企业把关的角色。可能对于一些人来讲只是一份工作，但是就企业来讲，无论他们怎么看待测试，他们依然将产品的质量的好坏直接挂钩到了测试身上。

测试岗位遍布各个行业，如果你只是在做移动互联网内的一个交互娱乐应用的话，可能责任还没有体现出来。但是还有很大一部分的人一直工作在银行、铁路、航空、医疗等领域，这些测试必须负责，他们关系到老百姓的生命安全。就如同《测试之美》中曾经提到，作者在几年前做的是医疗行业的测试，几年后自己母亲生病，维持着母亲生命的正是自己曾经测试过的医疗器械。只有当这个时候，自己的安心来自于自己的负责。所以我希望各个行业的测试们负起一份责任。

十、勇敢

正因为测试行业需要发展，测试技术需要进步，所以更加需要测试人去勇敢地钻研、尝试、实践、创新。很多测试人碍于自己只是一个打工的人，而不敢站在更高的角度看待问题；碍于自己内心的恐惧而看不起自己，觉得自己不是做技术，或者不是能够解决眼前问题的人；又或者碍于自己性格内向，而从停止了沟通前进的步伐。我曾经一直这样和我的员工说：“很多事情你不敢去做，很多事情你不知道怎么去做，但是不要忘记，你不做总有人会去做。他们做了所以他们变得有名有财富有知识。而你，还是你”。

可能能够让测试精彩的远远不止这些。我希望能够看完并且学习的人不仅仅是那些初入测试的人，更是那些有经验的测试盒管理层的人们。希望大家能够将这种精神传递下去，这样测试行业才会进步，才能有更多的人进来一起努力。

整篇文章都只是在说如何让测试更加精彩，那么测试怎么让生命精彩呢？段念段老师曾经在最后闪电演讲的时候说过：做测试的人，测试所带来的思想会慢慢地无意识地改变着自己的生活。很多的能力以及知识、态度都会在自己生活中起到很大的作用。其实就是这样的意思，我们作为一个人到世界上活上一回，既然只有一回，那么我们一定要活得够精彩，否则不就是太不值得了。而读者们往回看，这十点如果同样能够在你的生命中学到、深入，那么我相信你的生活、生命绝对会变得更精彩、更有意义。

C.3　中国人的纠结

——写于 2012-09-02 23:24:22

以下所有的举例均来自于生活中真实的例子。

国人纠结的地方太多，我相信很多人自己或多或少是这样的人，或者身边有这样的人。但是我们应该有什么态度去面对呢？我相信很多 80 后真的就是选择所谓的成熟、宽容。磨平了自己的棱角以为就是一种成功。但是在我看来，根本就是一种懦弱。虽然我希望更多人的能够看到客观，能够敢说、敢做，但是相反的，我也不希望看到更多的人和我一样，变成一种异类。

一、软弱

这种软弱体现在各个地方。假想着很多时候自己看到了路边的抢劫，看到了不道德的事情，有多少人会挺身而出？有多少为了工作而工作的人们，追求的只是那些钱，却不敢勇敢地追求自己想要的？有多少员工因为上级的一些评价而最终“承认”自己的错误。这种例子几乎比比皆是。

这种软弱会演变成另外一种现象“墙头草”或者更甚至“两面派”。在我看来，国

人不是没有主见，但是没有多少人敢说出来。主见就是一种主观判断的想法，这种想法可能是对的，也可能是错的。但是很多人就逆来顺受，软弱就变成了一种规矩，一种应该有的态度。

企业中，有多少软弱的人在为了钱工作着，数不胜数。人活着就那么一辈子，小学的时候可能每个人都有着非常远大的理想,但是现在呢？可能连工作上的问题都不敢提出，都一直忍受。慢慢地一辈子就这样过去了。你活出了什么？有自己的影子么?

二、不愿去反省

很多人会反抗，会和我说“我天天在反省，我反省得很彻底”。是的，你们都在反省。但是你们少数时候反省的是自己，基本上在反省别人。

从大的层面上来讲，反省不是一个老百姓应该做的事情。因为他们只要不犯罪，对得起自己的家人、朋友、自己。那么他们反省的事情基本上都是一些鸡毛蒜皮的事情。而真正该反省的是一些中层，上层的人员，但是他们往往是反省别人。之前看到在讨论一个观点，为什么富人还要不停地压榨穷人。一张图配得很好，一个富人挺着一个富态的大肚子，一个乞丐拿着破碗在这个富人的大肚子下面乞讨，这位富人却拿着望远镜在看，说：“哪里有穷人，我怎么看不到”。是的，富人的肚子太大，以至于已经将穷人遮住看不到了，形如蝼蚁。

企业中也是如此，当听到有客观的反馈的时候，比如一些工作量上的问题、待遇公平的问题。人员沟通协调的问题的时候，上层做出的反应大多会是一种装傻的态度。他们往往觉得没有问题，有问题的是你。所以你应该好好工作，不要胡思乱想。然后很多人就如同我第一点说的，继续懦弱地工作着。

社会中亦是如此，富人有富人的生活，穷人只能过穷人的生活。他们站的太高，根本看不到所谓的穷人。就如同以前的皇帝，老百姓受苦受难，皇帝哪怕再圣明也无法得知。这种流程倒是真正的保留到了现代的社会。社会影响企业，企业影响个人，上一代影响下一代。慢慢地，有权有地位的人总是对的，因为他们有钱有地位，而穷人永远是错的，不是因为他们穷，而是因为他们再也没有说出事实的底气和勇气。

千百年来，穷人天天在反省自己，而富人也天天在反省穷人。

三、结果导向

我相信这句话在企业中听到过的人不占少数吧，KPI 一个很可爱又可恨的东西。那么结果导向就真的那么有说服力么？我相信很多人心里有答案。

对于现代，很多古代的英雄事迹流传下来。因为什么？因为他们成功了。很多人会说，只有你成功了别人才能看得到你。这句话是废话，但是更加正确一点的说法应该是，只有你成功了，别人才能看到你的这次成功。因为只能代表你的一次成功，不能代表所有。但是国人现在基本都是反其道而行之。一次的成功代表了一个人的一切。甚至很多根本不能算所谓的成功。

当你和你的主管一起工作，你辛辛苦苦、任劳任怨。最终结果是你所做的全部功劳归你主管。你会怎么样的心情？很多人说，这样太不公平，我肯定会去和老板说，沟通、交涉。是的，但是事实可能远比这样复杂得多，我相信很多人也会忍气吞声。

假设一个男人，和女朋友谈恋爱，同居了之后在外面继续找女人，连续找了两年。最终和同居的那个女人结了婚。所有人都会认为，绝对的好男人，上海第一好男人。为什么？结果导向。可能很多人会说，别人的隐私我们怎么会知道？是的，很多人会这样认为不可怕，可怕的是这个男人自己也这样认为。

现在的孩子为什么一直要这个、要那个，要卖肾去换 iphone。为什么现在的攀比心理越来越强烈就是因为结果导向。人已经不再是人，他们根本不去追求好像是自己的钱去买来的，而只是去追求一个结果。你们看，我有我自豪。为什么？是的，社会教育的他们，企业教育的他们，家长教育的他们。

现在的家长为什么不要让孩子输在起跑线上。逼着孩子从小学这学那。国人不要批判自己没有创造力，这样的行为从小就磨灭了孩子的创造力，你们说将来怎么会有？家长为的是什么，为的是让孩子上一所好的大学。这样就一定有好的前途。好的大学和好的前途划等号吗？很多孩子在刚进大学的时候会去想我以后要做我喜欢的工作，要有所作为。但是结果呢，很多人去学了自己根本不感兴趣的东西，去考了不感兴趣的学校。为什么？结果导向，进入×××学校就能够找到好的工作。注意，这里是好的工作，而不是自己想要的工作。

这样的一切已经变质，结果导向只会让国人越来越迷失自己，自己一直在欺骗自己，永远不会再有真话，永远也没有人会追求过程，没有人再会注重细节。你让我造桥，一年我造好了；你让我修家电，我换零件修好了；你让我出钱供你读书，我去杀人、抢劫、卖淫，然后给你钱。这种变态不是别人强加的，而是我们的“创造”。

四、越老越好

在中国什么都是越老越好。老的都有资格，要尊老爱幼。工作经验久的就一定好。

我相信很多人会遇见，当今社会老人比谁都霸道。老人已经成为了一种定时炸弹。如果是一个正常的社会，老人上车不会听到“老人卡”，老人上车更不会比谁力气都大得往上挤，老人上车之后所有人也不会犹豫要不要让座。而现如今，一切都是相反的。我们还需要尊老吗？答案肯定是要。但是不是说是个老人就需要去尊重。我们是一个有思维的人，可以有判断性。

企业中很多人可能工作经验比较多，离职公司也比较大，然后就变成了一种好的象征。这种象征甚至可以让上层颠倒黑白。比如一个微软出来的员工，他可以去别的工作无所事事，然后拿着不公平的工资，过着像人一样的生活，而其他人只能过着畜生不如的生活。当你去和别人以及你的老板说明这样一个客观事实的时候，别人总会觉得那个人肯定不会像你口中说的那么一无是处。老板也会觉得你是在抱怨，为自己的工作找借口，看别人不爽才这样说的。一切的一切因为什么？因为那个人是微软出来的，因为他的工作经验多。如果角色反过来，估计那个无所事事的人早已经被开除。

国人的眼中就是如此，所以社会是现在的一副老态，年轻人也找不到一丝主见和活力。更多的只是服从和睁一只眼闭一只眼，得过且过的活着。

五、讲道德不讲道理

在中国，人情是个神奇的东西。关系好也是一个神奇的东西。不得不说，你在企业，在社会中碰见一些问题之后，你可以客观地评价，但是你不可以客观地去说出来。因为你必须要顾忌到所谓的道德，否则你会死得很惨。

现在的微博，问题越来越多。今天打空姐了，明天干露露了，后天吃出虫子了等等。但是讨论更多的不是道理，而是道德。道理是一种客观的存在，是对就是对的，是错就

是错的。而道德却存在于道理之上，让很多人不得不去欺骗自己，欺骗别人。比如 A 和 B 在讨论自己的衣服，A 问 B 好看不好看，B 觉得 A 是非常好的朋友，自然得说非常好看。但当 A 问了很多不相干的人的回答之后，A 才觉得 B 根本就是撒谎。其实这种事情非常多，但是可怕的却是 B 却不认为自己撒谎了，当 A 去责问的时候，B 还觉得非常的冤枉。

当你存在一个团队中，你的团队中有的人很负责，有的人很努力，团队的主管却无所事事。当你从客观的角度觉得有问题存在的时候，你只要不去提出，一切的平衡不会被打破。一旦你去和你的老板说了这样一个事实，那么问题来了。你的老板不会来和你讲道理，却会来和你讲道德。道德就是你不应该抱怨，不应该破坏团队气氛，你应该好好反省自己。这一切看似很好的劝解，背后回过头来想想，没有任何道理可言，并没有解决主管无所事事这样一个不争的事实。最终导致没有再会从客观事实去看待问题，也没有人敢说出客观的事实。大家都是靠关系，靠道德活着，不用讲道理。

六、磨平棱角，成熟的标志

这个也是我最想说的一点。成熟是什么？幸福是什么？很多人认为变得圆滑了，把很多事情看淡了就叫做成熟，就叫做处事不惊。这种举止已经变成了成熟的代名词。社会会变成现在的样子，就是因为中国人全部成熟了。是的，我相信熟的不能再熟了。医生开了过期的药，没事儿，反正照样能吃。一直在吃亏，也没事，吃亏是福嘛。一直活在不被看到的环境下，算了算了，过一天算一天，反正工资照拿。同事自杀了，伤心了，难过了，过去也就过去了。今天蒙冤的官司没有打赢，也算了，至少还活着。这种所谓的睁一只眼闭一只眼，得过且过都被叫做成熟。这种成熟在我看来只不过是不去看清客观事实，逃避客观事实的一种行为，一种懦弱、奴性、自欺欺人的行为！这样看来，幸福也就不难理解了。不要去计较，不要去争论，不要去看到问题的本质，得过且过就是一种幸福，活着就是一种幸福。国人更多地会把这种态度理解为宽容、成熟。实在是太荒唐了。

以上是读《中国人的纠结》读后感，一切的例子全采于身边的实例。请不要对号入座。一切只是那么简单罢了。有的人选择看开成熟，有的人选择无所谓，和自己没有关系。和我一样愿意去纠结客观事实的人们，我其实愿意相信，你们是没有出路的。请不

要向我学习。

C.4 黑盒不是白盒的绊脚石

其实写这篇文章的主旨就是希望广大的测试不要认为做了黑盒，做了无聊了或者做了几年之后就做白盒了，一直做黑盒没有进步什么的，千万不要认为黑盒就是白盒的垫脚石。黑盒和白盒的关系近似于 ST 和 ET 的关系。两者没有优劣之分，两者是互相支持的。并且不得不承认，现在大部分的测试人员是做产品测试、业务测试，纯逻辑类的毕竟少，而这类产品的测试更多的需要黑盒，其中所谓的自动化是为了更好地做黑盒才进行的，在我看来这个根本不是白盒测试。所以我也一直说自己是一个黑盒的测试人员，因为我的确没有做过白盒，或者说我从未放弃过黑盒。

这两天也有好几个人和我说："你做了也好久黑盒了，是差不多要做白盒了"。我是实在没有理解其中的逻辑关系。曾经在我的测试回忆中我也写到过，测试是一个很很神奇的过程。这个过程是一个压缩时间，创造可能出现的环境的过程，也就是在短时间内将最终用户在无限长的时间以及未知的使用环境下进行还原并查看软件的现象。这个就是软件测试的一种本质。而无论什么领域，从全局来讲，黑盒永远占着不可替代的位置。而所谓的技术，所谓的自动化，所谓的白盒就是让我们进行时间压缩和模拟环境的技术。并没有前后之分。做测试重要的是思想、随机应变的能力、测试用例的设计以及一颗不停探索的心。

C.5 测试需要反省

首先还是需要感谢雄立 xiongli 让我看到了她的几篇文章，包括她总结的 pdf。真的很中肯。并非偏向哪一方。能够在几年前有这样的感悟真的让我很佩服。

每个公司的测试都会有自己独到的方法进行测试，可能某些点代码测试比较好，可

能某些点功能测试比较好。这样的可能×××比较好其实很多，重要的是选用最好的方法去测试自己面前的产品，发挥自己最大的努力去学习，去做好。本质上又有何优劣。哪种合适，哪种效率高我们用哪种。不过唯一不变的就是测试的思维以及 TC 的设计方法。这个才是本质，也是需要深挖的点。

我这里为什么要写测试需要反省。每个人都需要反省。在作出任何评价前先审视自己。而我是测试，我就注重审视自己和这个行业。别人我管不到。

我还是一个初学者，我每天还是一直和广大的测试们进行交流，五湖四海的朋友们在交流。而我现在最大的一个感觉就是测试行业的人员，我们太浮躁了。我们太在意别人的眼光了。我们相比 IT 界对于测试的要求来讲太弱了。

测试、项目经理、产品、开发等所有的人目的就是使用自己特殊的技能来将产品做好，做到让用户满意，不是吗？不是大家来对打，来 pk 的。意义何在？就比如昨天的事件，我尊敬几位前辈，对我来讲都是前辈，都是有真材实料的实力的。但是很多的开发跳出来和我谈什么薪资，谈什么能力。这种意义何在？我就算承认测试的薪资不如你们，测试的能力不如你们，难道这个职业就消灭了么？大家讨论的初衷无非是在讨论怎么将测试做得更好，怎么将产品做得更好。而不是争谁牛逼，谁的钱多。我不得不在这里吐槽你们，太庸俗了！

但是反过来，正因为我一直和测试在交流着，我看到了测试人员的浮躁。每个人可能能力不同，每个人的天赋可能不同。但是看待事物的态度以及你生活的态度不能如此悲观。从我看来，很多的测试人员每天不想着提升自己的能力，每天不好好规划自己的职业，不停的在网上、在 QQ 群、微群等喊着“黑盒没有技术含量啊，自动化厉害啊，薪资好低啊，为什么开发一直看不起我”等等的。但是测试人员们，醒醒吧，我们要审视自己，自己真正给企业带来了多大的利益？请客观地审视你们抱怨的问题。测试为什么要分成多种类型，测试用例的设计方法为什么要有那么多，测试为什么要关注用户体验，为什么要关注需求。一切的一切是为了更好地测试。

说黑盒没有技术含量的测试们，你们有真正地学习过用例的设计方法么？你们有真正的了解黑盒测试是怎么测试的吗？如果就像你们说的，黑盒就是点点鼠标、屏幕的

话，不用别人说，我肯定会说，黑盒早就灭亡了。

说自动化牛逼，盲目追求自动化的测试们，你们有真正地经历过自动化么？你有真正地了解过自动化么？自动化只是仅仅写代码么？自动化要付出多少时间？多少人力？多少技术？是否需要企业的支持呢？是否自己所在的环境以及做的业务、产品是否真的适合自动化么？到底是适合 UI automation，还是适合 UT，还是适合 api 测试等。你们了解过么？不要天天喊着去做自动化。与其喊，不如自己去了解，去实践，这样才能够进步！

说薪资好低的测试们，你们真的了解低的本质吗？我真的无力吐槽了，薪资好低，最基本的本质就是你本身。能力和你的薪资不成正比么？如果是，那么肯定有成正比的地方你可以去，如果不是，那么你看看是谁的问题！

说别人看不起自己的测试们，说难听点。看不起总有道理的。你也可以看不起别人。你能够在各个点看不起别人。但是你少一块肉么？别人会少块肉么？你的将来是别人决定的么？你会每天少活一分钟么？不要 care 别人的眼光！ok！？

好吧，你们理解了么？可能很多人会说，这些人就不用去 care 了，他们根本不懂测试。他们慢慢会懂的。但是我要说的是，我看到的正是测试年轻的一代，正是测试新鲜的血液。如果认为他们不懂测试，我承认。但是不代表要放弃他们。现在很多的测试人员基本都是迷茫的状态。

其实有没有测试无所谓，有没有开发也无所谓。因为从理论上来讲，测试可以兼职开发做产品，开发也能够兼职测试做产品，我曾经还兼职 pm、market 等职业。其实目的就是一个，高效地完成项目，做好自己的产品和业务。

当然，很多人会吐槽我，说工作就是为了钱，为了生活，大家为了生计，何苦那么较真。那么我想说的是，每个人的追求不同。该文章只是写给热爱测试，热爱自己事业的人。至于别的，我管不了那么多。

PS：本篇是用来吐槽业界太多的浮躁的测试人员，虽然每个行业都有这样那样的人。但是感觉自己作为的行业这样浮躁的人特别多。本文绝对没有针对性，请勿过度吐槽。

C.6 《钝感力》有感——测试中的钝感力

感谢徐毅寄给我的《钝感力》和《编程高手箴言》两本书。当读到《钝感力》的最后几章的时候我才发现原来五六年前自己看过这本书的节选,突然感觉那是多么的有亲切感。

《钝感力》一书作者要告诉人们的不是应该怎么迟钝,而是做人之道、恋爱之道、夫妻之道,甚至长生之道。其中的例子全部选自作者自己的亲身经历,虽然很多例子人们觉得也不是非常大的事情,但是往往在我们生活中缺乏的就是作者所说的“钝感力”。

就测试而言也是一样的,在这样一个浮躁的社会,无论你是天才还是庸才,缺乏钝感力,那么你拥有的是你的过去,却无法掌控你的现在和未来了。

测试人员 A 和测试人员 B 都在同一家公司,两个人分别在两个项目中进行测试活动。项目中出现了这样、那样的问题,领导分别找了两个人谈话,小小批评了下。测试人员 A 在接下来的项目过程中也被同事嘲笑或者批评几句,慢慢地自己觉得自己做什么都不行,自卑感越来越重,在接下来的项目中也消极地对待同事以及工作,渐渐地整个项目漏洞百出。而测试人员 B 神经大条,觉得这次没有做好,下次做好就可以了。面对同事的嘲笑也当成学习的动力,并且总结自己的错误,用心投入了接下来的项目。最终测试人员 B 所在的项目取得巨大的成功。这种钝感力是测试人员所需要的,面对错误、面对失败不要太在意这个点,而是需要朝前看。

在技术飞速发展的今天,自己坚定方向才不会迷茫。测试人员 A 和测试人员 B 都是在行业中跌打滚爬了多年,他们都经常的去参加一些活动并且与其他同行的人员进行沟通。测试人员 A 在某次交流会中得知某技术 C 很不错,马上转而研究,又在某朋友交流会中得知朋友 D 做的自动化不错,马上又向 D 讨教学习自动化。时间一长,A 都没有深入了解,最终无法在任何一个行业中立足。而测试人员 B 坚持自己的选择,在看到别的新兴技术或者好的行业的时候,了解情况之后,结合自己现在的选择制定出了接下里的学习方案,一步一个脚印地走着属于自己的路,慢慢地也就成为了这个领域的大牛。我们没有必要对别人所做的,或者所在的行业垂涎欲滴,不要被这些所迷惑,而

丢失了自己。这样只会在回过头去看的时候发现自己碌碌无为。

对陌生人、对同事、对事业、对朋友、对妻子、对家人，对自己其实都一样。很多事情不要太过敏感，让自律神经放松一点，这样你会觉得世界更加广阔，生活更加美好，生命也更有意义。不是吗?

更多的博客文章可见：http://blog.sina.com.cn/u/1881320895

更多的移动技术可见：www.testerhome.com

读书笔记

后　记

如果你看到了这篇后记，那么说明你至少对我的故事是感兴趣的，或者你已经看完了全书等着吐槽我。无论是什么原因，我都由衷地说句：谢谢！。

人的一生中会碰见很多人，会发生很多变化，对我也是一样。虽然我刚工作没有几年，但需要感谢的人实在太多了，比如滔瑞的 CEO 周震漪、豆瓣的段念等。在这短短的从业 4 年期间，我认识了很多行业同仁，我和他们互帮互助的场景依然历历在目，我想告诉大家，我们的故事不仅会继续，还会更加精彩！

这本书本身就是一个故事，而这个故事能够顺利地和大家见面，我还需要感谢清华大学出版社的两位编辑：文开琪和栾大成。是你们让我的梦想成为了现实，让我的信念和思想真正地传递给了更多的人。

关于前言部分提到的故事如下（无论你看完是什么感受，都无法否认这是事实）：

故事要从我在专科读书开始说起。我出生在一个单亲家庭，母亲在我很小的时候，为了生计出远门打拼，直到我结婚前还是和外公外婆住在一起。家里最大的愿望就是我能够考上一所不错的大学。我还非常清楚的记得高考成绩公布的那天，我家人对我失望的态度，当然我自己也对我自己失望透顶，不过心里也清楚，原因是自己太喜欢玩电子游戏。相信过来的人都知道 CS、魔兽争霸、星际争霸，可以说各个平台的游戏我全都消耗过青春，最终的结果就是我进入了一所不错的专科学校中口碑最差的软件专业。

专科整整 3 年过得非常混乱：学校胡乱落后的课程安排，同学们每天吃喝玩乐的态度以及自己对于将来的恐惧和迷茫。一晃眼到了毕业季，往往很多事情就是那么巧合：恰巧我没有在一所好的大学、恰巧专科是三年全日制、恰巧我在 2009 年毕业了，又恰巧是金融危机爆发的一年。我依稀记得那时寝室的同学虽然表面都很淡定，但只要其中一个人接到了面试电话，那么大家心中马上就开始波涛汹涌了。2009 年我们软件专业的同学中，太多的人被迫去了自己不愿意去的公司和岗位，而我却因为一份所谓的软件

实习工作而改变了我之后的工作甚至人生的轨迹。

我去了一家科技公司做工程师，实习工资是一个月 500 元，工作内容是每天要外出进行 3 家证券大楼顶部卫星接收器的硬件维护。就这样，每天我在没有任何保险措施的情况下登上那些陌生大楼的楼顶，为他们做维护。其中有年久失修的旧宅，也有在市中心新建的办公大楼，无论是哪一种对我而言都充满着极大的挑战。那时我非常乐观地称呼自己为“现代蜘蛛侠”。就这样个月过去了，家里的老一辈非常担心我，怕我哪天出什么事故，而我因为性格原因以及抱着在金融危机的时候好不容易能够找到一份工作的处理由继续坚持着实习。

一件意想不到的事情发生了那是 2009 年的夏天，在实习的第 2 个月。有一天工作的时候，由于楼顶的水泥层年久失修我跌倒了。之后虽然自己强忍着伤痛继续上班，但膝盖上硕大的伤口却在一天一天地恶化，最终不得不在家静休。在家里的这段时间我问自己，这难道就是我想要的工作吗？这难道就是我的追求吗？我不知道自己到底想要什么样的工作，也许能够呆在办公室办公对当初的我而言已经是很幸福的了。经过深思熟虑之后，在快要转正的前几天我提出了离开公司的要求。

就这样我辞掉了在金融危机时依靠自己能力找到的唯一的一份工作，是的，我成了一个失业游民。接下来的故事也许很离奇，也许很曲折，但这就是人生，这就是生活，是我测试生涯的开始，也是新的人生的开始。

2009 年 10 月，加入了触宝科技，作为第一任正式的测试工程师。

触宝应该是我最为感谢的一家公司。作为自己的第一家公司实在觉得自己很幸运，他们教会了我太多的东西。在这里尤其感谢引荐人叶闻宇、CEO 王佳梁以及其他合伙人。

2011 年 2 月，在我团队中，一位和我年龄相仿的测试工程师因抑郁症去世。

创业公司的压力不言而喻，而压力在每个人身上的表现方式也很不同。我对测试事业的态度也是在这个时候改变的，这本书所想阐述的核心思想也是这个时候产生的。软件测试对于太多人而言太陌生，从那个时候开始，我下决心要尽全力去普及正确的测试观点给他人。

2012 年 10 月，正式在测试之道网站开始尝试免费分享网络课程。

这里要感谢第一届 ChinaTest，它是我打开人脉的重要会议，同样要感谢测试之道（http://uniontesting.com/），给了我这样一个陌生人重要的平台。现在大家依然能够看到在该网站上有很多我免费分享的课程，尤其是移动测试方面的。从这个时候开始出现了“Monkey”这个愿意分享的人的名字。

2012 年 10 月，我以个人名义创立了“移动测试会”公益测试沙龙组织。

也许很多在上海的移动互联网测试工程师都知道，当时我觉得其他地区的沙龙活动很多，尤其是北京，自然地，希望自己身边也有。也许是我自己的消息闭塞，最终决定与其等待，不如自己去创造，于是就创立了“移动测试会”。其实在创立过程中碰见很多没有预料到的困难，比如资金、场地、活动前期后期等的安排。我也是和很多公司的负责人交谈过，最终都以失败告终。在此非常感谢“创业公社”给我免费提供了场地，让我的移动测试会能够一次次顺利地展开。

2013 年，我孤身一人前往上海各个高校宣传软件测试。

在举办沙龙的过程中，我发现很多朋友对于软件测试的理解非常模糊，更多的是稀里糊涂地就做上了这个职业。我认定要改变这样的现状，应该更多从学校的学生入手，因而决定前往上海高校进行宣传。经过自己不断地交涉，成功地在上海大学、复旦大学、华东师范大学 3 所学校进行了宣讲。在此非常感谢华东师范大学的孙海英老师，她的一句“如果你想改变更多的人，出一本书的效率远远比沙龙、宣讲高得多。”让我下定决心写这本书。

之后的故事也许看上去变得那么顺理成章了。我担任了很多测试大会沙龙的讲师，也开始认识更多的人，很多人自我介绍的时候喜欢介绍自己公司的光环，自己学校的光环，而我就要打破这样的风气，靠的就是自己的这份激情和热情。曾经有太多的人质疑我，问我做了那么多，东奔西跑，搞沙龙自己还贴钱进去到底为了什么？我的初衷仅仅是为了能够影响更多的测试同仁，让大家的知识能够共享，从而一起进步，仅此而已。

整本书最后看起来就如同自己这 4 年工作的回忆，酸甜苦辣也算有滋有味。最后依然感谢这一路上我认识的朋友们，感谢帮助过我的伙伴们，当然最应该感谢的是你——本书的读者。